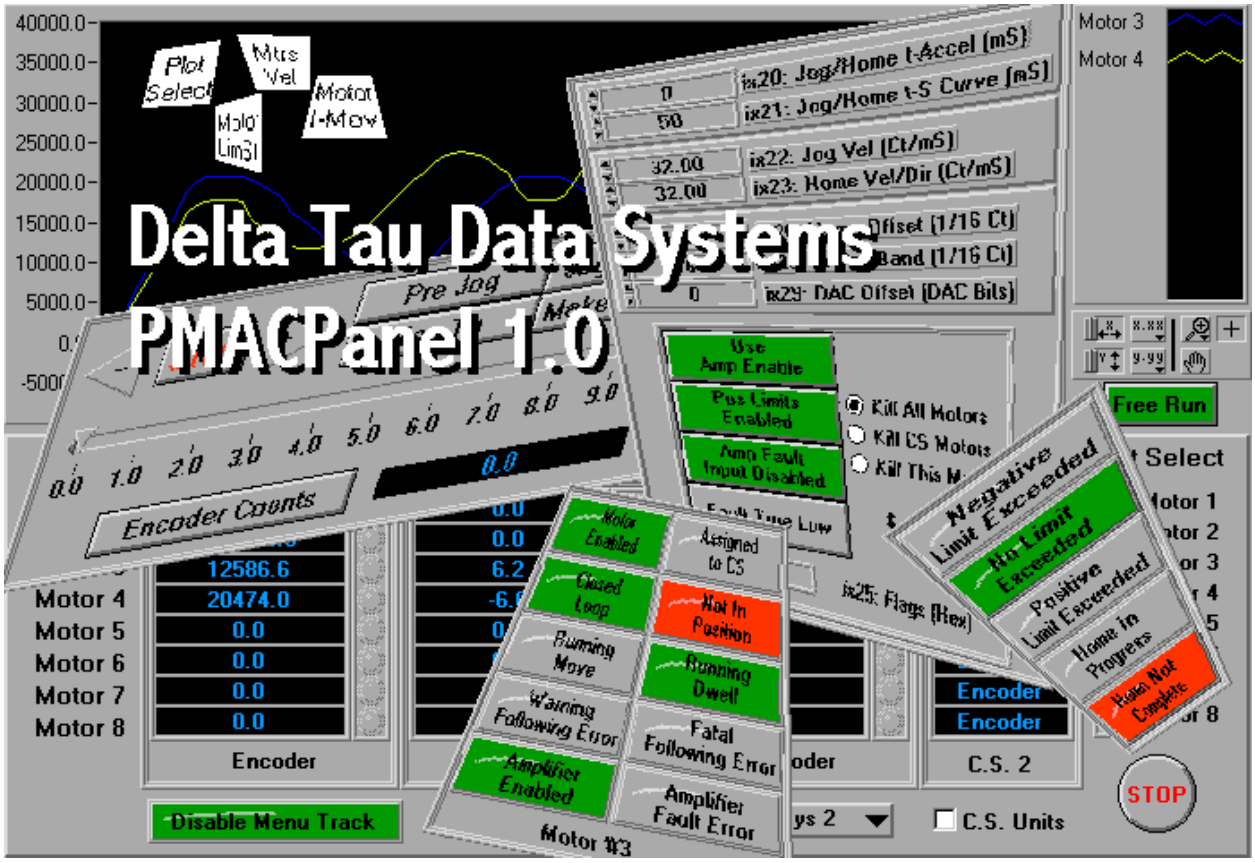
PMAC Motion Control for LabVIEW 5.0

PMACPanel User Manual



By Delta Tau Data Systems, Inc.

Contents

Chapter 1 - Overview 1

Introduction....1
Manual Layout....1
Organization....1
Conventions Used in This Manual....3
Use Caution When Running the Examples....3
Safety Summary....4
Motion Commands....4
Keep Away From Live Circuits....4
Live Circuit Contact Procedures....4
Electrostatic Sensitive Devices....5
HW Interfaces....5
Magnetic Media....5
Technical Support....5
By Telephone....5
By FAX and E-Mail....5
World Wide Web (WWW)....5
Bulletin Board Service (BBS)....5

Chapter 2 - Getting Started 7

Unpacking and Inspection....7
PMAC Compatibility....7
Customer-Furnished Hardware....7
Customer-Furnished Software....8
Delta Tau Software....8
National Instruments Software....8
Microsoft Software....9
PMAC Options for PMACPanel....9
Technical Documentation....9
PMACPanel and Your Computer's Display....10
Installing PMACPanel....10
PMACPanel Software....10
Configuring the Device Driver....11
Testing the Device Driver....13
Configuring LabVIEW....15
Installing the Release View....15
Creating Your Own View....16
Mass Compilation....16
On-Line Help....16
Configuring PMACPanel Communication....17
Testing PMACPanel Communication....18
PMAC Communication I-Variables....19
PComm32 Communication Buffers....20
Trouble Shooting PMACPanel Communication....20

Chapter 3 - PMACPanel Basics 22

PMACPanel and PMAC as Client and Server 22
Application Development Components 23
Pewin32 - PMAC Executive 24
PTalk - ActiveX Controls for Visual C++ and Visual Basic 24
PMACPanel - PMAC for LabVIEW 5.0 24
PMACPanel Interface to PComm32 25
Device Management 26
Query/Response Communication 26
LabVIEW and PMAC Numeric Data 27
Download Management 27
DPR Binary Data Buffers 28
PMACPanel Organization 28
Device Management and Communication 29
Query/Response Interface 29
Indicators, Controls, and VIs - ICVs 29
Motor ICVs 30
Coordinate System ICVs 30
Global ICVs 30
Accessory ICVs 30
Position Capture and Triggering ICVs 30
Program Development and Encapsulation Tools 31
Data Gathering and Graphical Tools 32
Code Interface Nodes and Dual Ported RAM 32
Sample Applications 32
Miscellaneous Utilities 33
Documentation 33

Chapter 4 - Application Basics 34

Basics 34
LabVIEW Techniques for PMACPanel 34
PMACPanel Indicator and Control Clusters 37
Accessing PMACPanel VIs 38
Clusters With an Associated Function VI 39
PMACPanel VI Terminal Conventions 40
PMACPanel Tutorials 41
PMACPanel Communication Tutorial 41
PmacTutor1- Accessing PComm32 42
PmacTutor2 - Sending Commands and Getting Responses 44
PmacTutor2a - Communication Logging 47
PmacTutor3 - Sending Commands Using Buttons 50
PmacTutor4 - Button and Response VIs 52
PmacTutor5 - Accessing PMAC Status 55
PmacTutor6 - Accessing PMAC I-Variables 57
PmacTutor6b - Accessing PMAC Memory 61
PMACPanel ICVs 65
On-line Commands 65
PmacMotor ICVs 68
PmacTutor7 - Position, Velocity, Error, and Jogging 68
PmacTutor8 - Motor Control with Status Monitoring 73
PmacTutor9 - Motor I-Variable Configuration 78
PmacMotors ICVs 81
PmacTutor10 - Requesting and Plotting Motor Motion 81
PmacGlobal ICVs 86
PmacTutor11 - Configuring PMAC's Global State 86
PmacCoord ICVs 90

PmacTutor12 - Using Coordinate System Definitions....91
PmacTutor13 - Configuring and Monitoring Coordinate Systems....94
PmacAcc ICVs....97
PmacTutor14 – Machine Input and Output....97
PmacTutor15 – ACC16D Control Panel....99

Chapter 5 - Development Tools 102

Basics....102
Tool Menus....103
Modifying the menu....104
Modifying PmacTerminalMenu....104
Basic Tool VI Requirements....105
Basic Tool VI Configuration....106
PmacTerminal....107
Basic Terminal 101....108
Basic Command Editing....109
Buffer Management....109
Terminal Indicators....110
Terminal Controls....110
Implementation Diagram....111
PmacTerminalJog....114
PmacTerminalEdit....115
Encapsulating Motion Programs....117
PmacTerminalExecute....118
PmacTerminalMotors....122
PmacTerminalMotorX-Y....124
PmacTerminalGather....126
Specifying Gather Addresses....131

Chapter 6 - Encapsulated Motion Programs and PQMs 136

Basics....136
PmacProgSubVI....136
PmacPQMExamp....139
PmacPQM Clusters....142
PmacPQM Conversions....143
PmacPQM Datalogging....144
Using Encapsulated Motion Programs....146
PmacTestExamp....147

Chapter 7 - Homing, Encoders, and Position Capture 151

Basics....151
Position Basics....152
Position-Capture....153
Trigger Condition....153
Homing....153
Action on Trigger....153
Home Complete....154
Home Position Offset....154
Zero-Move Homing....155
Homing Into a Limit Switch....155
Homing from PLC and Motion Programs....155
PmacHomeExamp....155
Configuring the Position Capture Trigger....157
Monitoring the Home Position Capture....160
Home Position Transformations....162

Encapsulated PLC Programs 164

Chapter 8 - Encoder Capture and Compare Operation 166

Basics 166
PmacEncoderPositionExamp 166
Encoder Position Transformations 167
Position-Capture for Non-Homing Purposes 169
PLC Capture Flag Processing 169
PmacEncoderCaptureExamp 170
External Triggers for Position Capture 172
PMAC Position Compare Operation 172
Required M-Variables 173
Pre-loading the Compare Position 173
Triggering External Action 175
PLC Compare Handling 175
PmacEncoderCompareExamp 176
Method 1 - PLC Operation 179
Method 2 - One-Shot Operation 180
Method 3 - PMACPanel Interval Generation 180
PmacEncoder Registers 180
Encoder Register Access 180

Chapter 9 - PMAC and NI-DAQ Interfacing 182

Basics 182
External PMAC Signals 182
Compare-Equals Outputs (JEQU) 183
Servo Clock (JRS232) 184
General Purpose Digital Inputs and Outputs 184
Synchronous M-Variables 185
Position Capture FLAGs 187
DAQ Signals 188
Analog I/O Channels 188
Trigger and Scan Clock Connections 188
PmacDAQMove 188
PMAC and AT-MI0-16 Signal Connections 189
Single Trigger DAQ 192
Multi-Trigger DAQ 192
Multi-Trigger DAQ with Servo Clock Sampling 193
Further Sampling Options 194
Other Interface Options 194

Chapter 10 - PComm32 Code Interface Nodes 195

Basics 195
LabVIEW Code Interface Node Basics 195
What is a CIN? 195
Using a CIN with PComm32 196
Setting up a PMACPanel CIN Configuration 196
Adding PComm32 Include Path 196
Adding Pmac.lib to Project 197
Configuring the IDE 197
The Easy Way to Add New Projects 197
Multiple CIN Projects in a Workspace 198
Creating a CIN C-Stub for PComm32 199

Chapter 11 - DPR - Dual Ported RAM 200

Basics 200
Required Background Understanding 201
General Architecture Notes 201
PmacDPRRealTime 202
PmacDPRRealTimeExample 202
PmacDPRRealTimeVectorExample 210
PmacDPRFixedBack 218
PmacDPRFixedBackExample 218
PmacDPRNumeric 221
DPR Addresses and Data Organization 222
PmacDPRNumericExample 222
PmacDPRNumericClusterExample 228
PmacDPRNumericCINClusterExample 231
PmacDPRNumericSlaveExample 233
PmacDPRVarBack 235
PmacDPRVarBackExample 236
PmacDPRVarBackVectorExample 240

Chapter 12 - Interrupts **241**

Basics 241
PmacInterruptExamp 241

Glossary of Terms **243**

Index **245**

Chapter 1 - Overview

Introduction

Congratulations on your selection of PMAC and PMACPanel for National Instruments LabVIEW, Delta Tau's complete motion control and instrumentation package. When you selected PMAC for the motion control portion of your DAQ application you gained far more than a simple positioning system. You get an integrated precision motion programming system with incredible capabilities.

With PMACPanel an entirely new world of motion control applications and capabilities opens. Motion that triggers acquisitions and responds to data gathered by SCXI, VXI, and industrial automation networks such as Device Net and Field Bus is now possible using LabVIEW's very popular and powerful graphical programming environment.

PMACPanel is an easily extensible set of more than 250 Virtual Instruments (VIs), Indicators, and Controls that allow you to communicate with and control PMAC from LabVIEW. It allows you to create a LabVIEW application that can monitor and control everything PMAC is doing using LabVIEW while at the same time preserving your understanding of the existing PMAC interface.

Manual Layout

This manual explains how to install and use PMACPanel to develop custom applications. It assumes the system integrator and PMACPanel developer has a basic understanding of the PMAC motion control board and LabVIEW. It does not cover the hardware and electrical configuration of PMAC or the use of Pewin32. If questions about a particular aspect of the installation arise, do not attempt the task until a thorough understanding is gained. Feel free to contact Delta Tau Data Systems, Inc., technical support at any time during installation. Refer to the Technical Support paragraph below for information on contacting our technical support department.

Organization

The manual is comprised of 13 chapters that take you through PMACPanel's many capabilities with installation instructions, architecture basic, tutorials,

terminal tools, motion program development, homing, capturing, triggering, interfacing and DPR.

Many of the chapters contain figures of the VI panels and diagrams to illustrate specific architectural approaches and VI implementations that you might need to modify to suit your purposes. Not all VIs are covered in detail. Many of them are complex and require in depth knowledge of PMAC's internal memory map that are way beyond a reasonable user manual. The last chapter contains detailed descriptions of the every cluster and VI in PMACPanel – even those not detailed in the previous chapters.

Chapter 1 – Overview

This chapter is an introduction to the PMACPanel features, manual layout, important safety issues, and how to access technical support.

Chapter 2 – Getting Started

This chapter specifies HW and SW equipment requirements, installation of the SW, configuration and testing of device drivers, and basic testing of PMACPanel communication.

Chapter 3 – PMACPanel Basics

This chapter introduces the client/server architecture that PMACPanel is based on and various issues involved in defining your application.

Chapter 4 – Application Basics

This chapter covers a number of LabVIEW techniques that are used in PMACPanel and may be new to you. This is followed by a set of tutorial exercises that start with opening communication with PMAC and start you down the path of developing your own PMAC enabled applications.

Chapter 5 – Development Tools

PMACPanel has numerous tools for developing and testing applications. These tools not only ease your development task; they are a great source for components and ideas of your applications.

Chapter 6 – Motion Program Interfaces

This is where you really harness the motion computer powers of PMAC for your applications. One of PMACPanel's nicest features is that it can create a wrapper VI around a native PMAC motion program that allows you to easily integrate the program into your application.

Chapter 7 –Homing, Encoders, and Position Capture:

When you use PMAC in a test and measurement application, establishment of an accurate home position is important. In this chapter we introduce VIs that enable you to do this from your application and introduce PMAC's HW position capture capabilities.

Chapter 8 – Encoder Capture and Compare Operation

This chapter introduces PMAC's ability to generate HW and SW triggers at specific motor positions. This includes a suite of VIs to configure these capabilities, and encapsulate PLC programs much like the wrappers introduced in Chapter 6.

Chapter 9 – PMAC and NI-DAQ Interfacing

In this chapter we show you how to tightly couple PMAC with standard NI-DAQ boards to establish, triggering conditions and if required, sample-by-sample registration of acquired data with motor positions.

Chapter 10 – PComm32 Code Interface Nodes

Accessing PMAC's DPR in an efficient way requires the use of LabVIEW Code Interface Nodes – CINs. CINs are compiled C code that a CIN calls when executing. In this chapter we discuss some basic issues associated with using Pcomm32 in CINs and develop an architecture for adding new CIN rapidly.

Chapter 11 – DPR - Dual Ported RAM

DPR allows PMAC to communicate with the host computer using memory in a way that significantly increases the speed of communication. In addition PMAC firmware has facilities that automatically copy predefined and user defined data into DPR for access by the host. In this chapter we introduce the VIs that support this interface.

Chapter 12 – Interrupts

PMAC supports the generation of interrupts from various sources. This chapter introduces one mechanism for handling PMAC interrupts from LabVIEW.

Chapter 13 – VI Reference

This is an exhaustive alphabetical reference defining the operation of every PMACPanel VI, control, and data type.

Conventions Used in This Manual

The following conventions are used throughout the manual:

<ENTER> *<CTRL+F4>*	Italic text inside arrows is used to represent keyboard keys or key combinations.
Edit»Edit Control	Dropdown menu selection or mouse operations.
`OPEN PROGRAM`	Mono-spaced is used for code listings.
PmacDevOpen	PMACPanel VI names
Bus Addresses	Arial bold text is used for dialog-box items.
`0x7FA000`	OCR text is used for dialog-box entries.
	Information which, if not observed, may cause serious injury or death.
	Information which, if not observed, may cause damage to equipment or software.
	A note concerning special functions or information of special interest.

Use Caution When Running the Examples

PMACPanel has many examples to introduce itself and verify things are working properly. You need to be aware of a few issues before actually running the examples.

- PMACPanel will cause your PMAC to execute motion. Please be careful.

- PMACPanel may require some changes in your PMAC's I-Variable configuration. You may also inadvertently change an I-Variable during the execution of some of the examples. If you have a currently working system please use Pewin32 to save the configuration **before** making changes to your PMAC.

PMACPanel will download programs and PLCs when some of its components run. If you currently have motion programs and PLCs that you value please use Pewin32 to save them before executing those examples that utilize encapsulated motion programs and PLCs. Otherwise, they will be replaced.

Safety Summary

The following are general safety precautions not related to any specific procedures and therefore may not appear elsewhere in this publication. These are recommended precautions that all personnel using PMAC must understand and apply during different phases of operation and maintenance.

Motion Commands

CAUTION *Until proper HW safeties have been installed, configured, and tested extreme caution must be exercised when moving motors to prevent damage and possible injury!*

PMAC moves motors. Sometimes very powerful motors driving sensitive equipment. PMACPanel developers are responsible for making certain they are thoroughly familiar with their mechanical setup, its capabilities, and performance and movement limitations. Furthermore that motion commands sent to PMAC do not cause damage or injury.

Keep Away From Live Circuits

Do not replace components or make adjustments inside equipment with power applied. Under certain conditions, dangerous potentials may exist when power has been turned off due to charges retained by capacitors. To avoid casualties, always remove power and discharge and ground a circuit before touching it.

Live Circuit Contact Procedures

Never attempt to remove a person from a live circuit with your bare hands. To do so is to risk sure and sudden death. If a person is connected to a live circuit, the following steps should be taken:

- Call for help immediately
- De-energize the circuit, if possible.
- Use a wood or fiberglass hot stick to pull the person free of the circuit.
- Apply cardiopulmonary resuscitation (CPR) if the person has stopped breathing or is in cardiac arrest.
- Obtain immediate medical assistance.

Electrostatic Sensitive Devices

Various circuit card assemblies and electronic components may be classified as Electrostatic Discharge (ESD) sensitive devices. Equipment manufacturers recommend handling all such components in accordance with standard ESD procedures. FAILURE TO DO SO MAY VOID YOUR WARRANTY.

HW Interfaces

When interfacing PMAC signals with any other data acquisition equipment be extremely careful to avoid shorting signals to supply or ground potentials. Furthermore, observe all signal load, voltage, and current limitations. FAILURE TO DO SO MAY VOID YOUR WARRANTY.

Magnetic Media

Motors and amplifiers may generate strong magnetic fields. Do not place or store magnetic media (tapes, discs, etc.) within ten feet of any magnetic field.

Technical Support

Delta Tau is happy to respond to any questions or concerns you have regarding PMACPanel. You can contact the Delta Tau Technical Support Staff by the following methods:

By Telephone

For immediate service, you can contact the Delta Tau Technical Support Staff by telephone Monday through Friday. Our support line hours and telephone numbers are listed below.

By FAX and E-Mail

You can FAX or E-Mail your request or problem to us overnight and we will deal with it the following business day. Our FAX numbers and E-Mail addresses are listed below. Please supply all pertinent equipment set-up information.

World Wide Web (WWW)

Delta Tau maintains a complete website containing many manuals, product updates, help files, application notes, and programming examples. We may be contacted at www.deltatau.com

Bulletin Board Service (BBS)

You can also leave messages on one of our Bulletin Board Services (BBS). The BBS is provided for our Customers, Distributors, Representatives, Integrators, et

al. We invite you to use this service. You can download & upload files and read posted bulletins and Delta Tau newsletters. Messages may be left for anyone who is a member/user of the Bulletin Board System(s). All you need is a modem and ProComm-Plus or similar communications program. Many Download-Upload Protocols such as Z-Modem are supported.

World Headquarters
Delta Tau Data Systems, Inc.
21314 Lassen Street
Chatsworth CA, 91311

Support Hot Line
Monday through Friday
8:30am to 4:30pm PST
Voice: (818) 998-2095
FAX: (818) 998-7807
BBS: (818) 407-4859
E-Mail: support@deltatau.com

Eastern U.S. Office
Delta Tau Data Systems, Inc.
10754 Decoursey Pike
Ryland Heights, KY 41015

Support Hot Line
Monday through Friday
8:30am to 4:30pm EST
Voice: (606) 356-0600
FAX: (606) 356-9910
BBS: (606) 356-6662
E-Mail: support@deltatau.com

European Office
Delta Tau Data Systems International
Industrieweg 175, Suite 7
3044 AS Rotterdam, Netherlands

Support Hot Line
Monday through Friday
8:00am to 4:00pm GMT
Voice: 31-10-462-7405
FAX: 31-10-245-0945
BBS: TBD
E-Mail: bradped@xs4all.nl

BBS Settings: Baud Rates: 1200 to 19.2
8 – data bits, 1 - stop bit, No Parity

Chapter 2 - Getting Started

Unpacking and Inspection

After receiving and opening the PMACPanel package, compare the contents to the packing list to ensure everything has been received. If anything shown on the packing list is missing, contact Delta Tau immediately. Carefully inspect all components for signs of physical damage.

PMACPanel consists of

- PMACPanel CD containing VIs, on-line documentation, and Microsoft Word 97' version of documentation.
- PMACPanel Technical Documentation Package

PMAC Compatibility

PMACPanel works with the following motion control boards

- PMAC-PC 4 or 8 axis
- PMAC-LITE 4-axis
- PMAC2 - 4 or 8 axis
- MiniPMAC

Installation and configuration of PMAC, amplifiers, and motors may have been performed by your system integrator or must be performed by you. Refer to the documentation provided by your integrator or with the purchase of your PMAC for details.

PMACPanel supports PMAC-2 with the exception of certain encoder specific capabilities such as encoder capture and compare.

Customer-Furnished Hardware

In order for the PMACPanel to operate, the following customer-furnished hardware is required:

- IBM or 100% compatible 486/66 MHz personal computer (PC). Pentium® or equivalent recommended.
- Minimum of 16MBof RAM. 32MB recommended
- A minimum of 100MB of free hard disk space.
- SVGA color monitor with minimum 1024x768 resolution.

In addition, the following optional National Instruments or third party data acquisition equipment may exist:

- Multi-function data acquisition I/O cards
- Signal conditioning equipment
- Image Acquisition
- GPIB Instrument Control
- Industrial Communication

Customer-Furnished Software

PMACPanel requires Microsoft Windows 95 or Windows NT 4.0 to operate.

Delta Tau Software

PMACPanel requires the existence of Delta Tau's PMAC device drivers. There are several possible options you may either have already installed or purchased with PMACPanel. The following information will help you determine where you are in the installation process and the steps needed to successfully install and configure PMACPanel

- You have PMAC Executive for Windows - Pewin32 installed and tested. You can skip those installation steps involved with the installation of the PComm32 device driver.
- You have PComm32 installed and tested. You can skip those installation steps involved with the installation of the PComm32 device driver.
- You have purchased and installed PTalk. You can skip those installation steps involved with the installation of the PComm32 device driver.
- You have purchased none of the above options. You will be directed to install a limited edition of the driver, configure it, and test it. Certain PMACPanel capabilities may not be supported without PComm32.

National Instruments Software

PMACPanel was developed for LabVIEW 5.0. Previous versions of LabVIEW are not supported by Delta Tau's PMACPanel motion package. LabVIEW must

be installed prior to installing PMACPanel. LabVIEW patch 5.0fix2, available from National Instruments via FTP is highly recommended

You may have other SW from National Instruments or Third Party sources such as NI-DAQ. There are no known conflicts with PMACPanel when using these packages.

Microsoft Software

Certain PMACPanel capabilities are implemented with compiled C code. These operate perfectly as is. Should you desire to modify them to suit your requirements or add other CINs for specific reasons you need Microsoft Visual C++ 5.0 or Microsoft Visual C++ 6.0. PMACPanel does not directly support other compilers. However, a talented SW engineer can make the required project modifications for other compilers in short order.

PMAC Options for PMACPanel

PMACPanel supports a wide range of PMAC's capabilities. Some of these require the purchase of additional Delta Tau hardware and software accessories. These options include:

- PMAC's Dual Ported RAM – This is required to utilize the PmacDPR collection of VIs.
- PComm32 – Complete PMAC device driver. PMACPanel provides a version of the complete driver with reduced capabilities. Tailoring of some PMACPanel capabilities may require the complete Pcomm32 release.
- Various PMAC I/O and accessory options

Technical Documentation

The *PMACPanel User Manual* included with this package is available in electronic form on the CD. The Microsoft Word document is located in the Documentation directory.

The *PMAC User Manual* and *PMAC Software Reference Manual* should have been provided with your PMAC. These references are absolutely necessary for any PMAC development effort.

If you will be modifying PMACPanel VIs that utilize DPR you need the *Pcomm32 Reference Manual*.

In addition to the required SW manuals the following technical manuals are required to successfully configure, installs and interface PMAC.

- Hardware manual specific to your PMAC model
- Manuals for PMAC options such as Dual Ported RAM

If any of these manuals are missing, please contact Delta Tau for a replacement before attempting your development.

PMACPanel and Your Computer's Display

PMACPanel's indicators and controls are configured for display on a computer with 1024x768 resolution or greater. You should set your display's resolution to at least this size to use them. PMACPanel VIs work at smaller display sizes but the panels will not fit within your display area. You can choose to resize the panel controls or change the size of your display.

Installing PMACPanel

Because of the number of SW drivers and steps in the communication process it is extremely important that each step be done carefully and tested before proceeding to the next. There are a few steps that must be taken prior to installing PMACPanel.

- Install, configure, and test LabVIEW 5.0 or greater and any patches.
- Install, configure, and test any National Instruments boards
- Install, configure, and test your systems' PMAC hardware

Install, configure, and test PComm32, Pewin32, and/or PTalk if purchased. If these options were not purchased you will install a limited edition of the PComm32 device driver included with PMACPanel.

PMACPanel Software

Before installing PMACPanel, read the license agreement included in this manual (behind title page). You should also check the release notes included with the manual and located in the Documentation directory on the CD for last minute changes. Installation of PMACPanel is done in two steps. First, the drivers must be installed. Next, the PMACPanel SW must be installed.

Installation of the Driver

Skip this step if you have already installed and tested PeWin32, PComm32, or PTalk. If you have not purchased one of the tools locate the directory PMACPanel Drivers on the CD and run Setup. The installation program will suggest a directory path where the program files should be copied. The suggested directory location is **c:\Program Files\Delta Tau\PMACPanel**. This will install the drivers and two applications - MotionExe and PMACTest.

Installation of PMACPanel

To install PMACPanel, locate the directory PMACPanel on the CD and run Setup. If you have properly installed LabVIEW the installation program will add several components to your LabVIEW installation. If Setup cannot locate LabVIEW specify its location or exit the installation and install LabVIEW. The primary PMACPanel component is the directory **PMACPanel.lib** in your LabVIEW installation directory. The library directory contains numerous sub-directories to organize the VIs, utilities, and documentation.

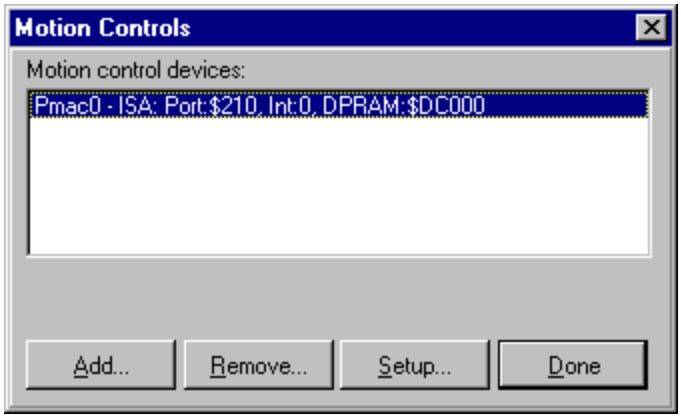
Configuring the Device Driver

You must configure and test the driver installation before running PMACPanel for the first time. If you have purchased and installed Pewin32 or PComm32 this step has already been completed. Proceed to *Configuring LabVIEW*.

PMAC communication configuration has been centralized in your operating system, making the setup of PMAC much like other devices in your computer (i.e. video card, sound card etc.). All setup is done through the "MOTION CONTROLS" applet accessible through your operating systems CONTROL PANEL or the included configuration program MotionExe. Before running this application it is important that all applications that use Delta Tau's 32-bit communication driver PComm32 be shut down. This includes Pewin32, NC for Windows, or any applications developed with PComm32 or PTalk.

To configure PMAC communication, click on the Motion Control gear icon

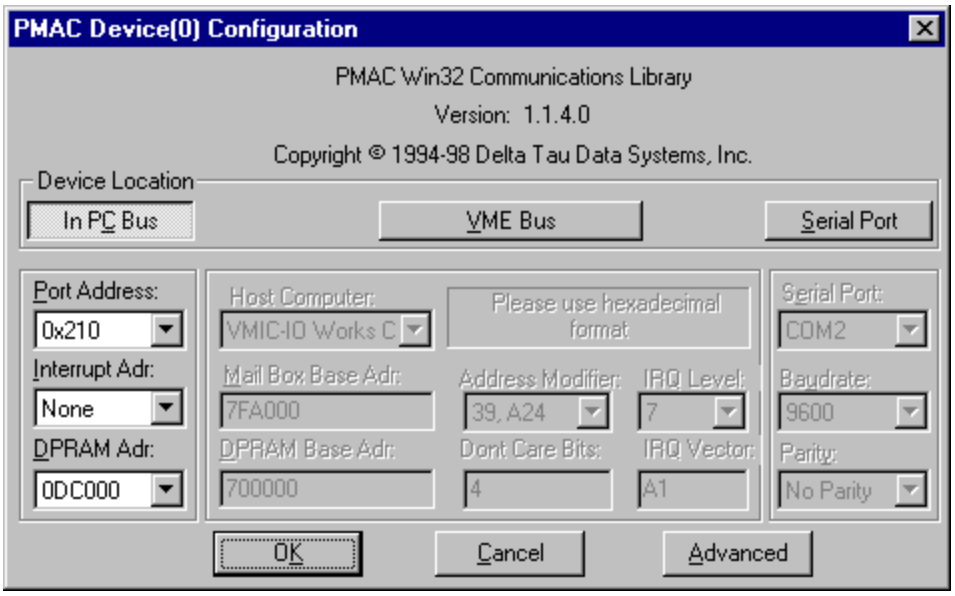
or execute the program MotionExe created during the installation. The following dialog box will appear



The Windows NT version of this dialog has extra buttons labeled "Unload", "Load", and "Startup". "Load" and "Unload" should only be used when trouble shooting the PMAC installation. "Startup" may be used to tell Windows NT how to load the PComm32 communication driver.

If this is your first time running the applet there will be no PMAC's listed in the "Motion control devices" list box. This is because none have been "Added" to your operating system yet. To add a PMAC press the "Add" button to get the following dialog box:

This dialog box prompts you for a device number to associate with the PMAC you are adding. Always start with your first PMAC as Device 0, the second PMAC in your system as Device 1 and so on. The applet will handle the enumerating for you. Press OK, to get the configuration dialog



This is where you specify how PMAC is connected to your system and the resources used by PMAC. The configuration you define here must match the hardware jumper settings on the PMAC itself and not conflict with those already assigned by Windows 95 or Windows NT to other system devices or National Instruments accessories. Serial port communication requires the use of a serial cable.

When the configuration is properly specified click OK. If PMAC is communicating with the driver properly the following dialog appears.

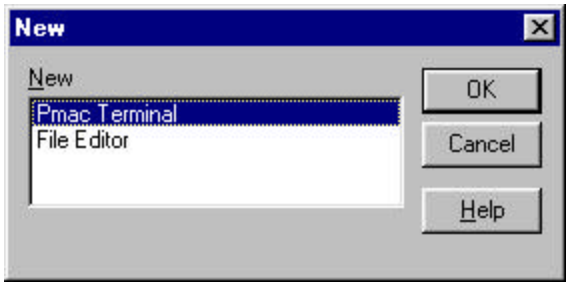
If there is a problem with PMAC, the assigned resources, or the driver the following dialog will appear.

To remedy the situation, check for resource conflicts with other devices in inconsistent hardware jumpers. If the problem persists contact Delta Tau Technical support.

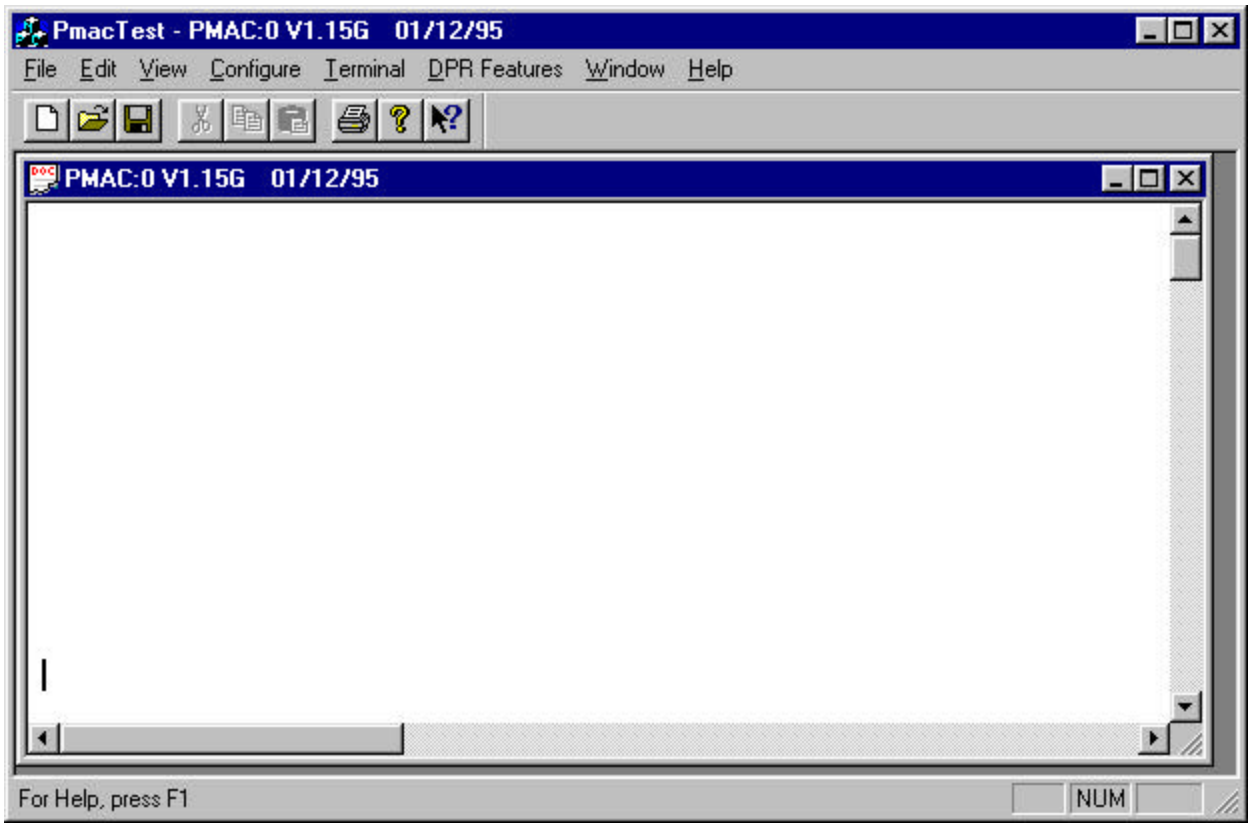
The "Advanced" button is used to configure DPR settings typically used with the Delta Tau NC for Windows software.

Testing the Device Driver

Initial testing of PMAC and the device driver are done with the program PMACTest included with PMACPanel or Pcomm32 drivers. When PMACTest executes, the following dialog appears requesting the preferred operational mode.



Click “OK” and a terminal window will appear


PmacTest - PMAC:0 V1.15G 01/12/95
File Edit View Configure Terminal DPR Features Window Help
PMAC:0 V1.15G 01/12/95
For Help, press F1
NUM

CAUTION *Until proper HW safeties have been installed, configured, **AND** tested **extreme** caution must be exercised when moving motors to prevent damage and possible injury!*

PMACTest is now in terminal emulation mode, allowing you to interact directly with PMAC. Although it is tempting to move motors when communication is first established in this step you should be **thoroughly** familiar with your mechanical setup and be certain that commands executed from PMACTest will not cause damage or injury.

Check to see if you get a response by typing I10<*Enter*>. PMAC should respond with a six or seven digit number. Now type III<*Enter*> -- PMAC should respond with a beep, signifying an unrecognized command. Next, satisfy yourself that you can communicate with the PMAC card at a basic level. Type P<*Enter*>. This requests a position. PMAC should respond with a number.

If your system contains a PMAC-LITE, you will still get back eight numbers when you type <CONTROL-F>.

Now type <*CONTROL-F*>. You should get back eight numbers (one for each axis) since <*CONTROL-F*> requests following errors from all eight motors; some or all may be zero. Please note that even with encoder counts as read-out (no scaling), PMAC's position is displayed with fractional counts.

If error dialogs appear or the responses are not as specified check PMACTest's help capability. It might help to revisit the previous section *Configuring the Driver*. If the problem persists contact Delta Tau Technical support.

Congratulations! You have successfully installed PMAC and the PComm32 communication drivers on your system.

Configuring LabVIEW

During the installation of PMACPanel the directory PMACPanel.lib containing the PMACPanel distribution VIs was created in your LabVIEW directory. There are three things that need to be done to seamlessly integrate PMACPanel into your LabVIEW development environment

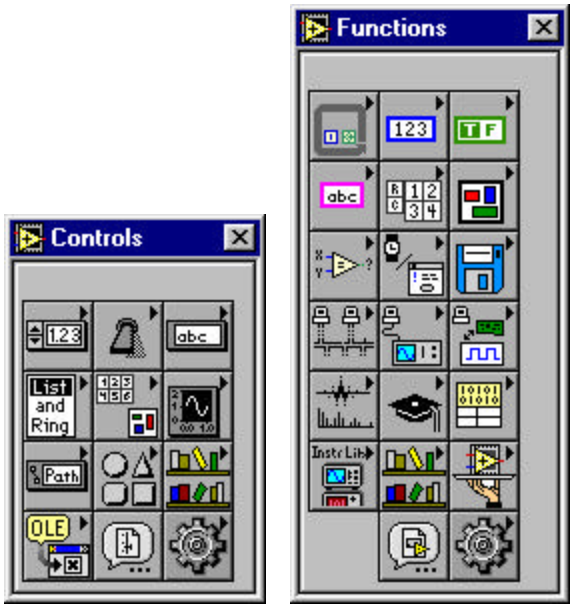
- Create a view
- Mass compile the VIs
- Configure PMACPanel for your PMAC driver configuration

Installing the Release View

To facilitate your use of PMACPanel you should install the view contained in the release or create your own view so that the PMACPanel VIs and controls are easily accessible from the Controls and Functions palettes and do not clutter your User.lib directory. The procedure for doing this is outlined here.

- Run LabVIEW
- Select **Edit»Select Palette Set»PMACPanel**. This view is a modification of the default view. If no PMACPanel selection appears check for a directory named PMACPanel in the directory LabVIEW\Menus.

The Controls and Functions palettes will appear as



Access to the entire suite of PMACPanel controls and function VIs is now available using the

icon and its sub-palettes. PMACPanel icons are text based and indicate the PMACPanel.lib subdirectory they are located in and their specific function.

Creating Your Own View

If you did not install PMACPanel in PMACPanel.lib or you already have a custom view to accommodate other LabVIEW packages refer to the LabVIEW manuals or Online Reference under the topic **Palettes Editor** to add PMACPanel to your palettes. This can be done by using **Edit»Edit Control & Function Palettes…** option, creating a new view, inserting a Submenu, and linking it to a directory - **PMACPanel.lib** (or your own name). The icon **\PmacDocument\PmacPanelIcon.bmp** can be added during the palette editing process.

Mass Compilation

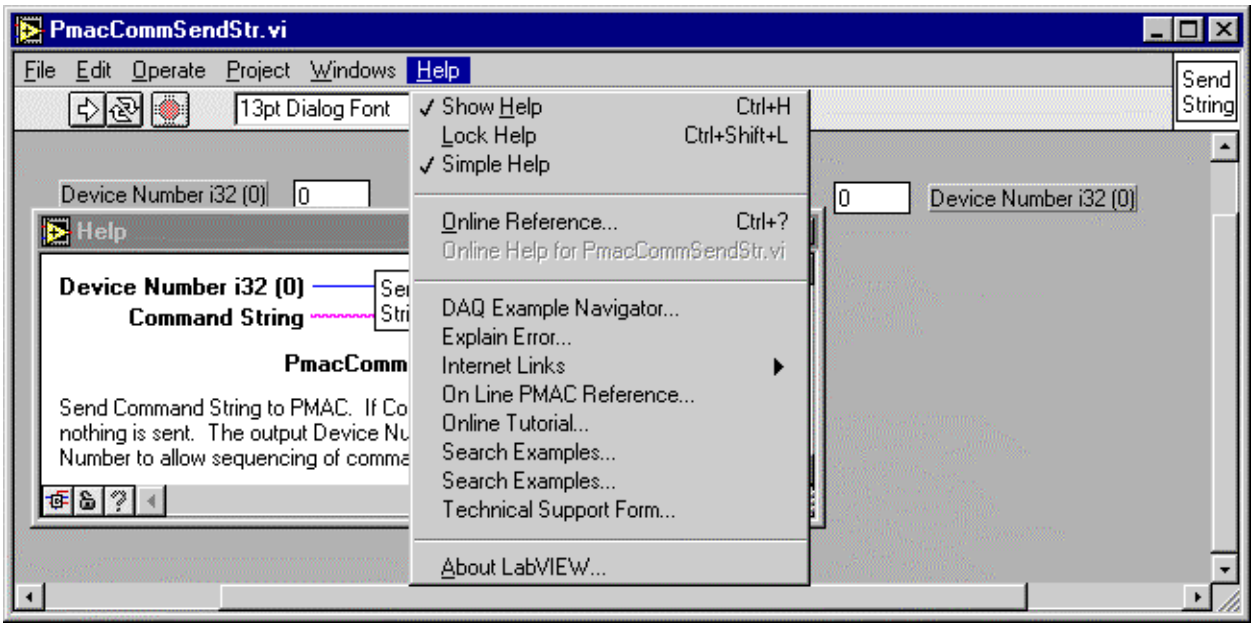
This step compiles the entire PMACPanel release so that there are fewer searches when loading VIs and confirms that everything can be found. Select **Edit»Mass Compile** to display the file selection dialog while you have a VI, any VI, open. Browse your way to the directory PMACPanel.lib and click "Select Cur Dir". LabVIEW will then begin loading and compiling the entire PMACPanel release. When this is complete click "Cancel".

If you have purchased the PComm32 package you have the ability to develop LabVIEW Code Interface Nodes that may require re-compilation See Chapter 10 for details.

If the compilation process encounters problems note the error message. The most common problem encountered will be its inability to locate the PComm32 driver extension Pmac.dll installed by the Pewin32, PComm32, PTalk, or PMACPanel. This should be located in c:\Windows\System. If the file is not in this directory try to locate it and determine where the file was placed. You can copy the file into the correct directory and reattempt the compilation.

On-Line Help

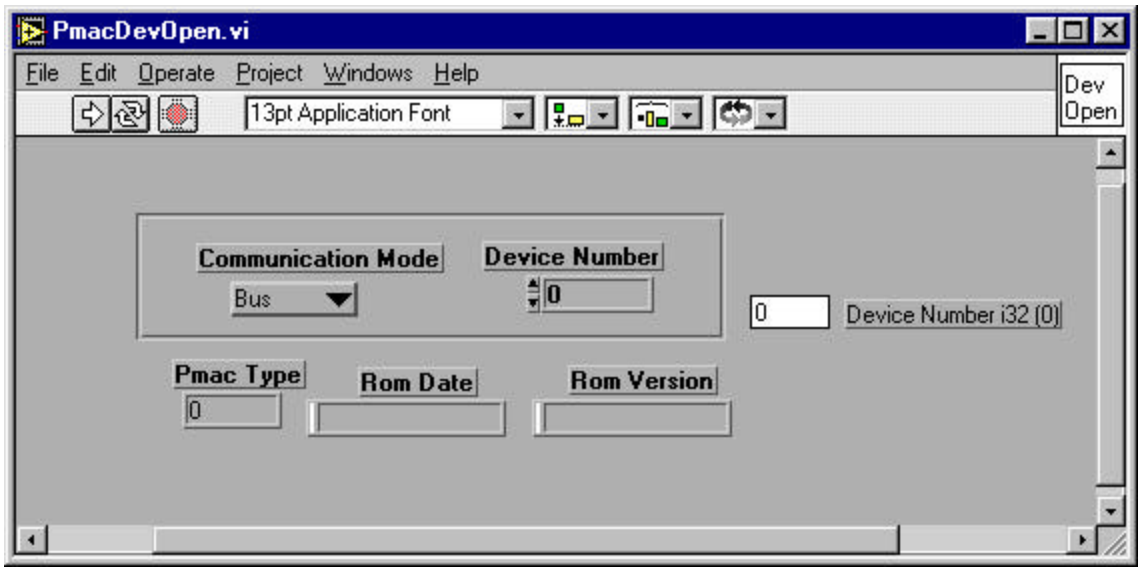
PMACPanel has extensive on-line help in two forms. Extensive documentation of every VI and its terminals is available using the standard LabVIEW **Help»Show Help** option. There are also several standard Windows on-line help files located in the **LabVIEW\Help** sub-directory. These include help versions of the printed manuals and several PMAC manuals. This is useful when examining the examples and tutorials. When LabVIEW starts the contents of this directory are parsed and any help files located in the directory are added to LabVIEW's pull down help menu. The figure below shows both forms of help. Note the entry in the pull down menu for *On Line PMAC Reference*. We will add other on-line help files as called for. Many of these can be down loaded from Delta Tau's web site and copied into the **LabVIEW\Help** directory.



Configuring PMACPanel Communication

PMACPanel communicates with PMAC using the PComm32 device driver configured previously. To access the driver from PMACPanel the device number and communication mode defined for the driver must be defined for PMACPanel. PMACPanel's primary device driver, **PmacDevOpen**, is configured with the following procedure

- Run LabVIEW
- Open the VI **PmacDevOpen**. Select **File»Open** and navigate to: **PMACPanel.lib\PmacDevice\PmacDevOpen.vi**. The following panel should appear

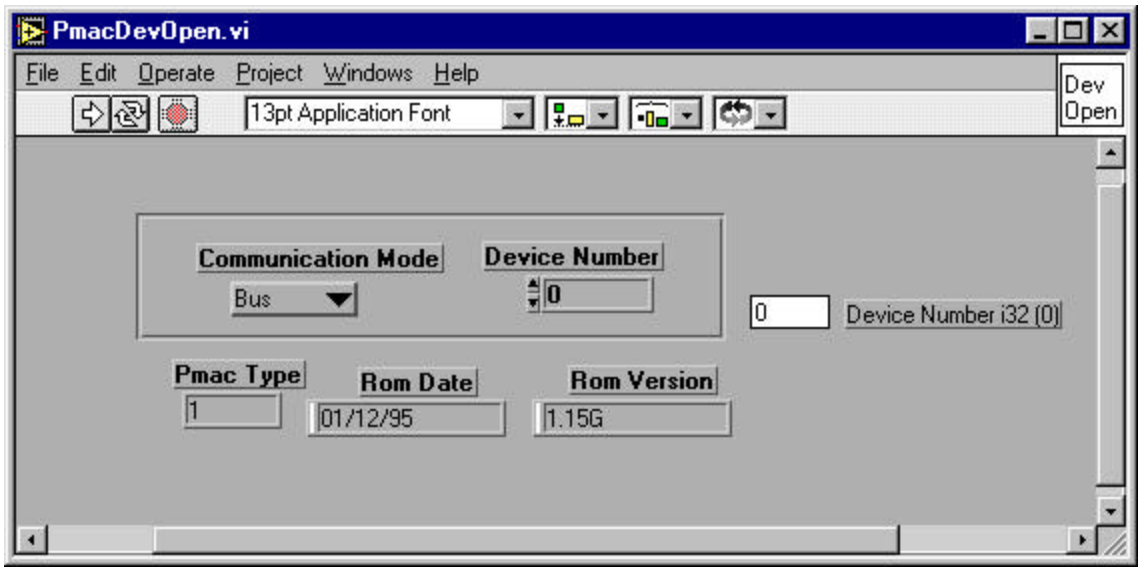


Set the Device Number control - not the indicator in the white box - on the front panel to the device number specified for your PMAC during the configuration of the driver. The default Device Number in a new PMACPanel package is 0. If this is your device no change is required. Otherwise, mo dify the control and make the value permanent using the right mouse button and the **Data Operations»Make Current Value Default** option then saving the VI.

The device driver manager allows you to select Serial Port or In PC Bus as the desired communication mode. The same selection should appear in the Communication Mode drop list on the front panel. The default mode specifies the use of the Bus. If this is your mode no change is required. If the desired communication mode is DPR the device driver control panel should specify In PC Bus along with a valid DPR address. The Communication Mode in this panel should display DPR. As with the device number this should be made permanent by using the right mouse button and **Data Operations»Make Current Value Default** option and saving the VI.

Testing PMACPanel Communication

The final step in the installation of PMACPanel is to test its ability to communicate with the device driver. After configuring **PmacDevOpen** and saving the default changes execute it using the run button on the menu bar. The panel should change to reflect the Type, Rom Date, and Rom Version of your PMAC as shown here.



If you see something like this on your panel - Congratulations! You have successfully installed PMAC and the PComm32 communication drivers on your system. Proceed to Chapter 3.

PMAC Communication I-Variables

PComm32 supports communication with PMAC using UNICODE and standard C/C++ ASCII strings. PMACPanel is configured to use C/C++ ASCII strings not UNICODE due to the use of LabVIEW's Call Library VI to interface LabVIEW with PComm32.

PMAC uses I-Variables to define communication characteristics. These are important to verify because the behavior of PMACPanel depends on their value. If PMACPanel communication is not operating properly configure these values using Pewin32 or the PMACTest application included with PMACPanel.

I1 - Serial Port Mode

This parameter specifies whether PMAC will use hardware flow control using CS and whether a software card address is required with each command. At present no software card address is required hence I1 should have a value of 0 or 1. If desired, a card address can be pre-pended to all communication with some modifications to **PmacCommSendStr**, **PmacCommGetStr**, and **PmacCommRespStr**.

I3 - I/O Handshake Control

This parameter determines what characters, if any, PMAC uses to delimit a transmitted line, and whether PMAC issues an acknowledgment (handshake) in response to a command. The preferred setting is I3 = 2.

I4 - Communication Integrity Mode

This parameter allows PMAC to compute checksums for the communication bytes sent between PComm32 and PMAC. This value should be I4 = 0 because PMACPanel does not currently add or strip a checksum.

I6 - Error Reporting Mode

This parameter specifies how PMAC reports command line errors. The preferred setting is I6 = 1. In this mode PMAC errors are properly parsed by PMACPanel and reported to the user with a pop-up dialog.

I58 - DPRAM ASCII Communication Enable

This parameter enables or disables the DPRAM ASCII communications function. When I58 = 1 PMACPanel sends and receives communication through DPRAM. When I58 = 0 communication is done via the Bus or Serial Port. Enabling ASCII communication is not required to access DPRAM available using other PMACPanel capabilities. Using DPRAM ASCII communication modifies I3 and will effectively disable PMACPanel's ability to properly parse error messages. The preferred value is I58 = 0. This implies that Bus or Serial is the preferred Communication Mode to be specified for **PmacDevOpen**. This has little impact on overall performance and does not preclude the use of DPRAM for memory mapped purposes.

PComm32 Communication Buffers

PMAC handles commands and responses in a very simple manner. PMAC commands that generate responses place the responses in an internal buffer that is transferred into the caller's buffer. If the entire response does not fit in the caller's buffer the data is held in PMAC until the remainder of the buffer is fetched. New commands sent to PMAC flush the response buffer prior to executing the new command. Hence, responses that are not fully retrieved are lost.

Communication with PMAC via PComm32 requires empty buffers into which responses are placed. THIS IS VERY IMPORTANT. The empty response buffers for **PmacCommGetStr** and **PmacCommRespStr** are created as 128 byte buffers. If larger default buffers are desired the size of the buffer can be increased to 256 - NO MORE. PComm32 internals cannot handle buffers larger than this. PMACPanel handles larger response buffers internally using **PmacCommGetBuffer**.

Trouble Shooting PMACPanel Communication

At this point it is assumed that the driver was successfully configured and tested as outlined in *Configuring the Device Driver* and *Testing the Device Driver*. If you skipped these steps revisit them. If you have Pewin32 or Pcomm32 and they work the problem is with your configuration of **PmacDevOpen** or the communication I-Variables.

- In the event that LabVIEW crashes when running **PmacDevOpen** reboot the computer to eliminate any damage to the driver and memory caused by the crash.

- Verify the correct operation of the device driver by checking the configuration using MotionExe or the Control Panel Applet. When you select "OK" from the setup dialog the driver attempts to contact PMAC and reports the success or failure of the attempt.

- Check the driver operation communication with PMACTest or Pewin32.

- Revisit the driver configuration and make certain that the device number and communication modes specified match those specified for **PmacDevOpen**. Make changes to the VI and retest the communication by running the VI again.

It is known that very early versions of Windows 95 do not work well with LabVIEW and the PComm32 device driver. If the problem persists contact Delta Tau Technical support. If the system continues to crash try to note any error messages in detail.

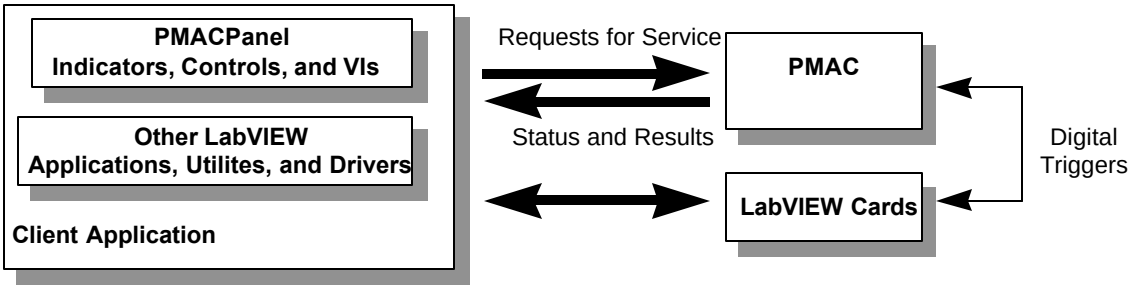
Chapter 3 - PMACPanel Basics

PMACPanel and PMAC as Client and Server

PMACPanel is a powerful LabVIEW toolkit that allows you to develop GUI based clients requiring precision multi-axis motion that integrate PMAC's unique capabilities with other LabVIEW devices.

PMACPanel is not intended to replace a thorough understanding of PMAC's powerful motion and PLC capabilities, its architecture, its command language, or its programming language. PMACPanel is not a Graphical Motion Language (GML) that allows you to write PMAC programs by stringing together a set of motion description VIs.

The client/server architecture, illustrated below, works exceedingly well with LabVIEW's G-Code data flow execution model and matches the model used to communicate with GPIB, Industrial Automation networks, etc. PMACPanel applications place requests to PMAC to set motion characteristics, run motion programs, and configure and monitor PMAC status. PMAC executes these requests without the assistance of the client application as long as power is applied. Using this model, PMACPanel applications can use data from other LabVIEW devices to control the motion and coordinate control of those devices with the execution of the motion.

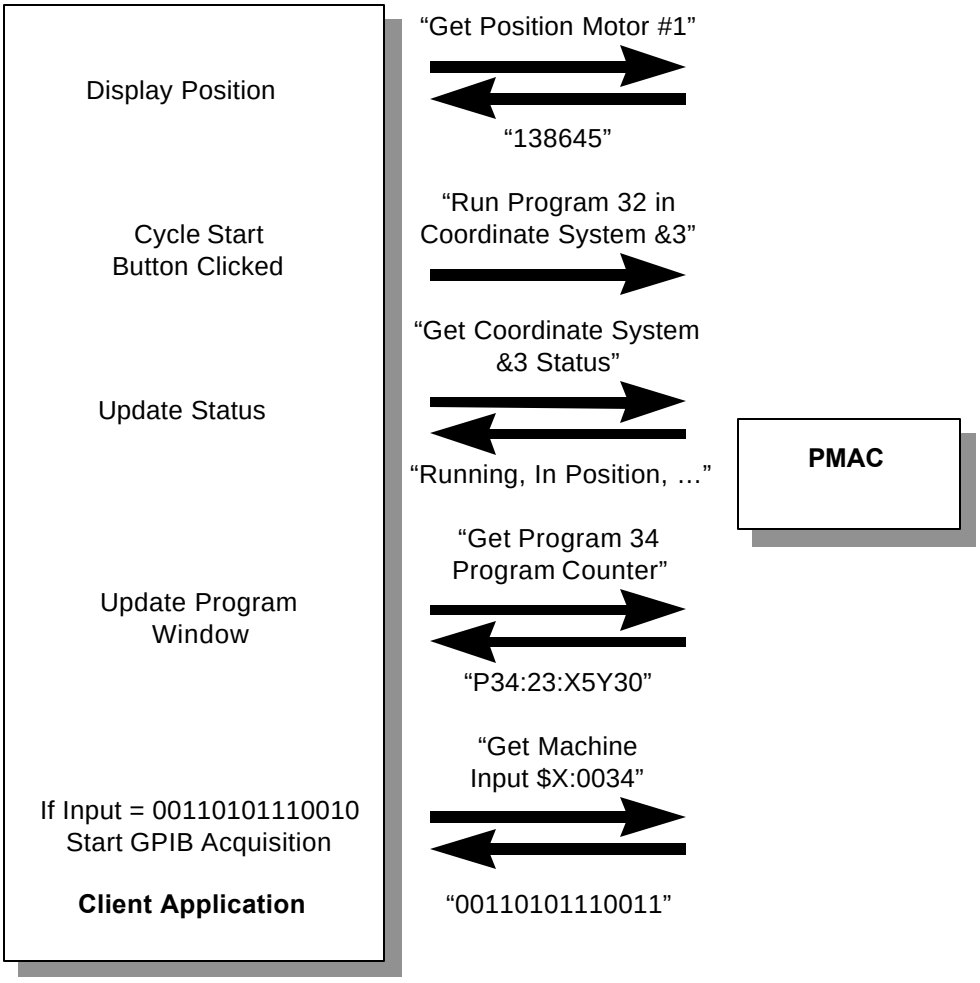


The figure below illustrates a set of Query/Response exchanges between the PMACPanel client application and PMAC. Commands for service are sent by the application in response to

- button clicks
- events such as measurements made by other instruments

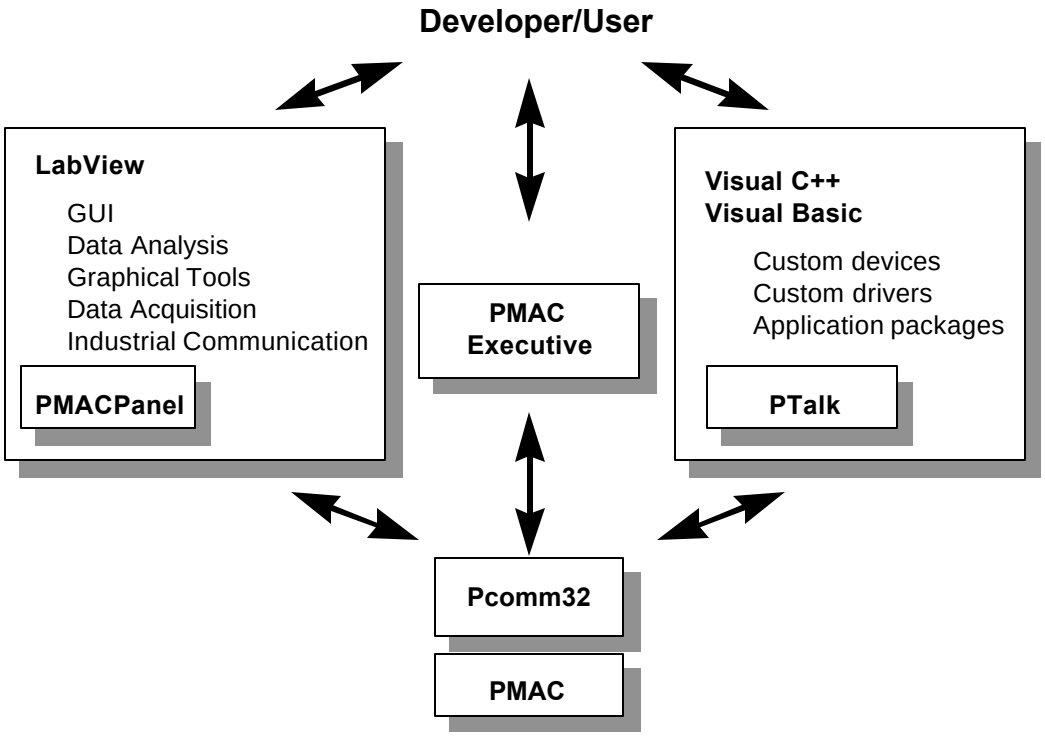
- polling requirements

In response to these requests PMAC responds with the requested operation or data. Because of this architecture, everything PMAC can do can be done under control of and in coordination with client applications.



Application Development Components

Depending on your motion requirements, system integration requirements, and performance needs there are three methods for developing your PMACPanel applications. The following figure shows the three primary components available from Delta Tau for your system development.



Pewin32 - PMAC Executive

PMAC system setup and configuration operations such as motion program development, motor tuning, limit and safety configuration, etc. are best accomplished using the executive. Once a set of sample motion programs is available a PMACPanel application can be developed. PMAC can still operate as a standalone controller. When you wish to interact with PMAC your PMACPanel applications are available to change the operation of PMAC.

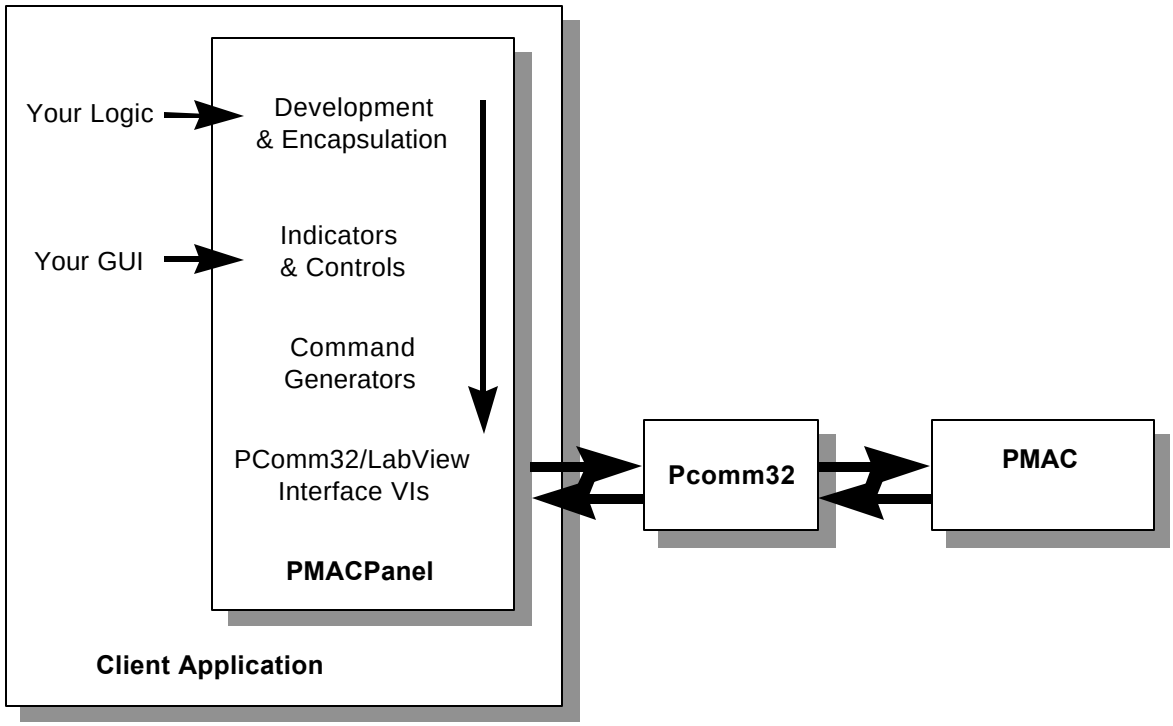
PTalk - ActiveX Controls for Visual C++ and Visual Basic

Applications with demanding computational needs can written in C++ and their needs can be communicated to PMAC using PTalk. As an ActiveX control PTalk can be used from within PMACPanel. It is also possible to compile this code into a dynamic link library or as a LabVIEW Code Interface Node and use it from your own custom VIs in a PMACPanel application.

PMACPanel - PMAC for LabVIEW 5.0

PMACPanel provides a complete suite of VIs to simplify and standardize your application's access to PMAC from LabVIEW. It is divided into 4 basic levels or categories. At the very lowest level are VIs that provide an interface to PComm32. Above this are two levels that provide collections of indicators, controls, and VIs that you can use in your LabVIEW applications to control PMAC and monitor its status. Finally, there is a level that provides program development utilities that can be used to encapsulate PMAC motion and PLC

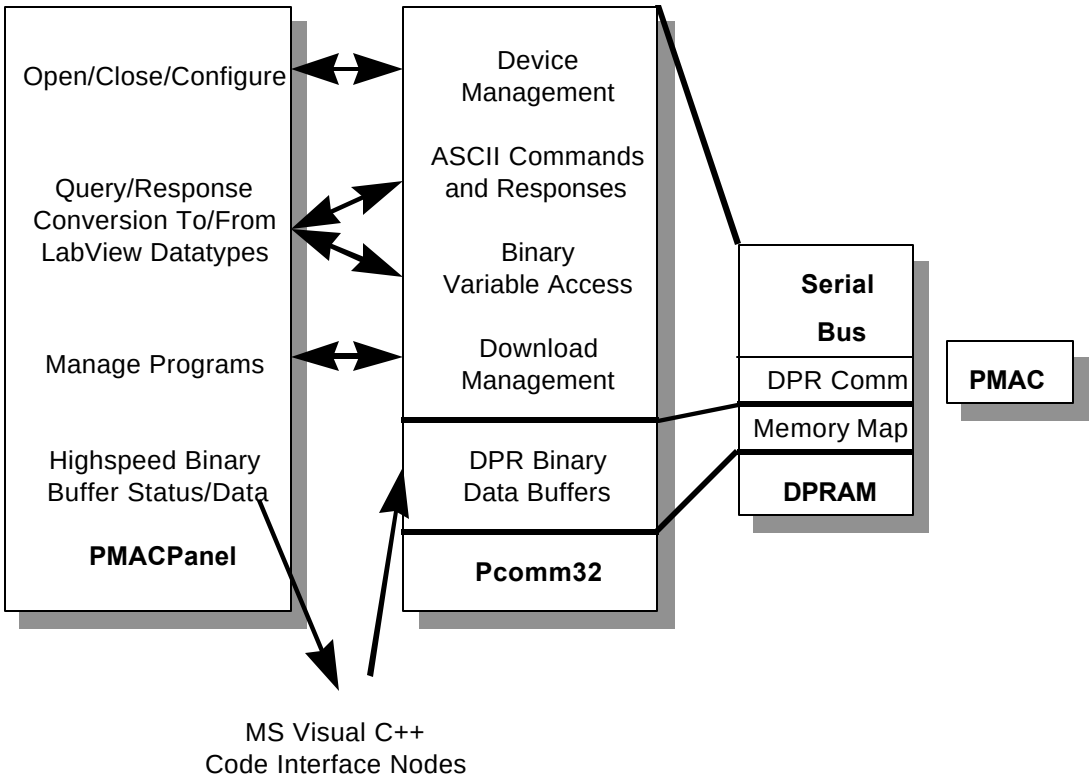
programs as VIs that are controlled by the application. In general, you will use the top three levels for application development.



PMACPanel Interface to PComm32

PMACPanel's lowest level provides a flexible set of capabilities through which all command interactions with PMAC is conducted. To do this PMACPanel wraps selected PComm32 functions with appropriate logic to provide a transparent interface between your LabVIEW application and PMAC. It provides three primary categories of access to PMAC. These are:

- Device Management
- Query/Response ASCII and Binary Communication
- Download Management



Device Management

DPR is primarily intended to pass real-time data gathering buffers and binary data for operations like inverse-kinematics between PMAC and the client. It is not required for general Query/Response interaction.

Access to the device driver is handled with the PmacDevOpen VI configured earlier. It defines whether Query/ Response communication uses the Serial Port, Bus, or DPR. Pcomm32 hides all details associated with a given mode from the user. If your PMAC is inserted into your host computer Bus communication is a sure bet. DPR, while slightly faster, is not required. If your PMAC is located remotely, then you need to communicate with it using the serial port. If your application does a lot of polling for status data you will see a marked decrease in application execution performance.

Query/Response Communication

Query/Response communication is the most basic form of communication with PMAC. This mechanism allows your application to use the entire set of PMAC's on-line Commands to interact with PMAC. The quickest way to build a PMACPanel application to control and monitor PMAC is to locate the functionality you desire in the *PMAC User Manual* and *PMAC Software Reference Manual*. You can then test it with Pewin32 or the **PmacTerminal** contained in PMACPanel. When you are certain you know what you want you, build a LabVIEW VI and select an appropriate PMACPanel VI to send that command and/or data to PMAC.

LabVIEW and PMAC Numeric Data

LabVIEW supports a wide range of native data types that need to be communicated to PMAC. This data must be formatted for transmission to PMAC and converted from PMAC responses into LabVIEW types. PMAC returns numeric data as decimal or hexadecimal ASCII strings. The query/response VIs convert this ASCII data into native LabVIEW numbers for manipulation and display.

PMAC uses the Motorola 56K series of Digital Signal Processors for its computational engine. The memory architecture is based on a 48-bit long word comprised of two 24-bit words. This organization allows one to access the entire 48-bit long word, or either of its two 24-bit words using the same address.

The long word at memory location $23F8 is accessed using D:$23F8 or L:$23F8. The upper word is designated X:$23F8 and the lower word is designated Y:$23F8.

PMAC's firmware uses several data types depending on the quantity being represented: 1 bit booleans, 8, 16, 24, 32, and 48 bit integers, and 24, 32, and 48 bit floating point numbers. Of these most integers are 24 or 48 bits. This presents an immediate issue that **must** be understood when developing your PMACPanel application. LabVIEW's short integers are 16-bits - too short for PMAC's 24-bit integers. Its long integers are 32-bits - too short for PMAC's 48-bit integers and too long for PMAC's 24 bit integers. Furthermore, LabVIEW does not support a 48-bit integer or 48-bit floating-point number. Fortunately, because Query/Response communication is conducted with ASCII strings the size of PMAC's data is somewhat hidden.

To simplify the interface between LabVIEW and PMAC, PMACPanel supports the conversion of PMAC data into one of six LabVIEW data types. Conversion into unsigned representations is easily done using LabVIEW's many conversion VIs. The following table enumerates the representations and names supported in each domain. The PMACPanel names Short, Ushort, Long, and ULong are used in the names of many Query Interface VIs.

PMAC	LabVIEW	PMACPanel
1 bit binary	Boolean	Boolean or Bool
1, 8, 16 bit integers	16 bit integer (i16/u16)	Short (i16)/UShort (u16)
16, 24, 32 bit integers	32 bit integer (i32/u32)	Long (i32)/ULong (u32)
24 bit floating point, 48 bit integers, 48 bit floating point doubles	64 bit floating point doubles	Double or Dbl

There are some situations, depending on the value of I9, where PMAC's ASCII response strings are hexadecimal not decimal. The conversion of these ASCII responses into equivalent native LabVIEW binary representations are handled by classes of VIs that use PComm32's binary variable access capabilities.

Download Management

Maintenance of PMAC programs is provided by a collection of VIs that directly access PComm32's download functions. These compile ASCII PMAC programs and download them to PMAC for execution.

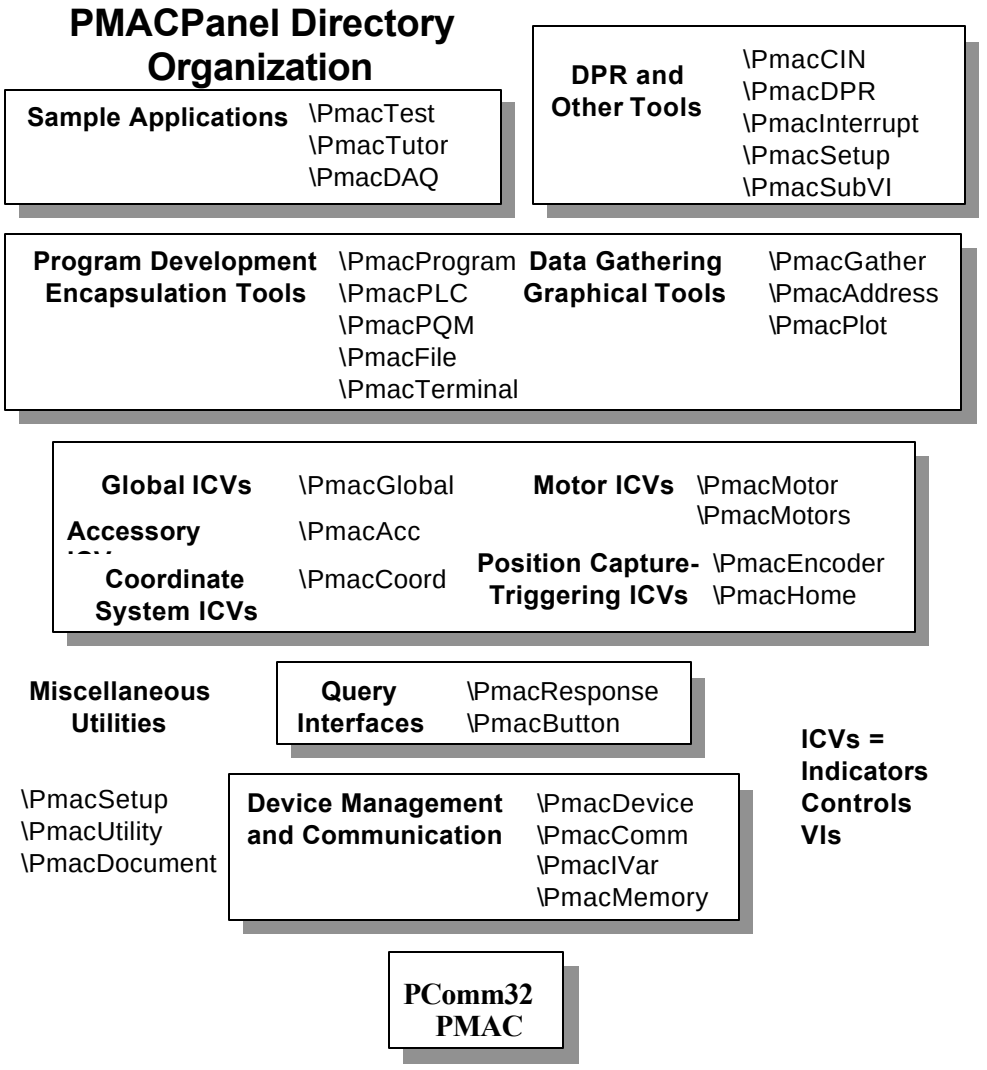
DPR Binary Data Buffers

PR represents a unique PMAC capability that rapidly transfers binary numerical data between the host and PMAC. This communication mode eliminates the string formatting and parsing required with ASCII communication. It is however, not the best solution for all problems. Chapter 11 covers the use of these mechanisms fully.

PMACPanel Organization

This brief explanation of PMACPanel's organization will help you to get the information you need quickly and painlessly as well as help you plan your application's architecture.

The PMACPanel library, contained in the directory PMACPanel.lib, is divided into five basic categories as illustrated in the following figure. These categories provide an increasing level of capability as you progress from the lower levels to the higher levels.



Within each category are several sub-directories containing collections of indicators, controls, and VIs (ICVs) that provide capabilities to make your application development task easier. The VIs in each sub-directory follow a naming convention that makes them easier to locate. For example, in the directory **\PmacCoord** all VIs are named **PmacCoord ….** The names of VIs in a given sub-directory may have shortened names to make them a little easier to fit on screens and such. For example, VIs in **\PmacResponse** are shortened to **PmacResp….** The collections of VIs in each category implement most commonly used capabilities and generally have examples to demonstrate their use. The examples will have the word **Examp** or **Example** at the end of their name. The remaining chapters cover numerous tutorial exercises and examples to demonstrate their use.

Device Management and Communication

At the lowest level of the architecture are VIs that configure and manage PMACPanel communication with PMAC using PComm32.

\PmacDevice - Configure and manage access to PMAC using PComm32. Configuration of this access must match the driver configuration set using the control panel applet or MotionExe application.

\PmacComm - Provide the ability to send on-line ASCII command strings and data to PMAC and receive ASCII responses from PMAC. Error handling and the ability to buffer communication for analysis and future reference are provided at this level.

\PmacIVar - Provide direct binary access to PMAC I-Variables. This collection of VIs provides an easy to use architecture for accessing I-Variables and avoids formatting problems that can arise when querying PMAC I-Variables that might be returned as hexadecimal values.

\PmacMemory- Access to PMAC's memory mapped variables and registers are simplified with this group of VIs.

Query/Response Interface

PMAC's client/server interaction model works by sending requests to PMAC and awaiting responses. Interaction with PMAC at this level removes much of the tedium involved in sending commands and formatting responses.

\PmacResponse - Send formatted command strings to PMAC and convert ASCII response strings into numerical values for use in you PMACPanel application.

\PmacButton - In most user interfaces queries are sent to PMAC as the result of Boolean user actions and conditions. This group of VIs send command strings to request responses when an associated LabVIEW Boolean condition is TRUE.

Indicators, Controls, and VIs - ICVs

This collection of directories and VIs provide an extensive set of indicators, controls and VIs that will tremendously speed the development of PMACPanel applications. Each of the 5 groups in this category provide prepackaged indicators and controls to manipulate PMAC's I-Variable setup, monitor execution status, and send commands to PMAC using appealing clusters of LabVIEW controls. Capabilities not implemented by these VIs can easily be

added by modifying the provided VIs. Numerous examples are provided to demonstrate the capabilities of the collection and can used as is.

Motor ICVs

Interaction with individual and sets of motors is provided by these collections of VIs. The collections contain VIs to request PMAC status and motor states and display their data on pre-defined cluster indicators. In addition, controls to Jog and control motors are provided to simplify testing and development of programs.

\PmacMotor - Monitor and control individual motors. There are numerous ICVs to get motor position, velocity, and following error, modify motor I-Variables, process motor status, and jog motors.

\PmacMotors - Monitor and plot the motion of collections of motors in defined coordinate systems. Plotting tools for selecting which motors and motion variables to plot are available. Samples of real-time strip charts and XY charts are provided.

Coordinate System ICVs

\PmacCoord - Monitor the execution of programs and definition of motor coordinate systems. This information is required for the development of user interfaces in which information is entered and displayed in coordinate system units rather than encoder units.

Global ICVs

\PmacGlobal - General PMAC setup and configuration are provided by this collection for VIs. These capabilities are used for the development of supervisory VIs.

Accessory ICVs

\PmacAcc - PMAC has numerous accessories that are used to provide digital I/O, analog I/O, and control capabilities. This includes the ACC16D front panel and the alphanumeric display. This collection provides basic ICVs that access the memory mapped data used by these accessories.

Position Capture and Triggering ICVs

PMAC has the ability to capture exact motion positions in response to external triggers and generate triggers when specified positions are reached. This capability can, in conjunction with National Instruments DAQ boards, be used to capture a position or trigger instrument control in response to specified positions.

\PmacHome – This collection of VIs provides access to the I-Variables and memory variables required for defining homing criteria and determining home position offset.

\PmacEncoder – This collection provides VIs to access and control the encoder gate array for position capture and compare. There are also tools for developing background PLC programs for generating Compare-Equal outputs.

Program Development and Encapsulation Tools

Developing PLC and motion programs that work seamlessly with PMACPanel requires four major components.

- Tools to edit and create programs
- Tools to monitor the execution and debugging of programs
- Tools to develop PMACPanel panels to communicate program specific data between the program and user
- Tools to encapsulate programs, their execution, and monitoring into a single easy to use VI

The purpose of these collections is to allow the integration of motion programs into a PMACPanel application. Either Pewin32 or the PMACPanel application tools documented in Chapter 5 can be used for the development of the raw motion programs.

\PmacTerminal - Terminal application tools for interactively creating, controlling, and monitoring PMAC and your programs.

\PmacFile - Tools for maintaining ASCII program files, LabVIEW datalog files, and downloading files to PMAC.

\PmacPQM - PMAC program execution is parametrically specified using P, Q, and M variables. For example, the number of times a move is executed, the increment of a move, or the radius of a circular move can all be specified using Ps and Qs. Specific machine inputs and outputs, and internal registers are accessible using M-variables. Mapping of these quantities to LabVIEW controls is facilitated by the ICVs in this collection. In addition, the ability to log this information to a LabVIEW datalog file and re-execute the motion at a later time is provided.

\PmacPLC - PLC programs and their execution status can be edited and controlled using the VIs in this collection.

\PmacProgram - This collection provides tools at a variety of levels to edit, download, debug, monitor, and encapsulate motion programs. Encapsulated programs

- Load themselves when executed
- Know their coordinate system and program number
- Can be executed by the click of a button
- Indicate the state of their execution
- Can be modified, monitored, and debugged from a powerful front panel
- Accept P, Q, and M variant data types from the **\PmacPQM** tools.

\PmacSubVI – This directory contains the actual encapsulated program and PLC wrapper template VIs.

Data Gathering and Graphical Tools

One of PMAC's most intriguing capabilities is its ability to synchronously gather a variety of motion data during the execution of a program or move. An example is the gathering of actual and desired position in response to step inputs. This data can be used to analyze the performance of a specific move or machine configuration using LabVIEW's powerful analysis suite.

\PmacGather - This collection of VI's allow the user to select motion variables to control the collection and conversion of data into standard LabVIEW analysis formats. Collected data can be output to files in tab delimited format for export to programs such as Matlab or Excel.

\PmacAddress - A collection of tools for specifying addresses, scale factors, and descriptions for gathering. Variables not already accessible from PMACPanel can easily be added to the tables.

\PmacPlot - A few generally useful plotting VIs for setting plot colors and legends. An XY Chart buffer is available to make an XY plot into an XY strip chart.

Code Interface Nodes and Dual Ported RAM

PMAC Dual Ported RAM (DPR) provides a high-speed binary data transfer mechanism that greatly speeds access to certain types of motion data. To facilitate this capability PMACPanel utilizes LabVIEW Code Interface Nodes (CINs). This collection of VIs demonstrates how to use this capability.

\PmacCIN – A detailed description of the process required for configuring your environment to use CINs with PMACPanel.

\PmacDPR – A large collection of VIs for configuring and operating the many modes of DPR supported by PMAC. There are numerous examples demonstrating the use of DPR and how to modify the supplied collection to suit your purposes.

\PmacInc – A directory of include files required to use CINs or modify PmacDPR. This directory will be empty if you did not purchase the PComm32 package.

Sample Applications

This collection of VIs demonstrates a general PMACPanel application and a set of tutorials to walk you through the correct use of PMACPanel capabilities.

\PmacTest - An all encompassing demonstration of program encapsulation and monitoring with 4 different motion programs, their PQM configuration, and real-time strip chart monitoring.

\PmacTutorial - A sequence of exercises covered in Chapter 4. These exercises introduce you to the basic architecture and proper use of PMACPanel in your own applications. All first time users of PMACPanel should read this chapter and examine the tutorial VIs.

\PmacDAQ – This collection of VIs utilize standard LabVIEW analog input DAQ example VIs and a PMACPanel motion VI to demonstrate a few of the techniques you can use to integrate PMAC and NI-DAQ boards to develop precision motion based data acquisition applications.

Miscellaneous Utilities

This collection provides many VIs used to implement PMACPanel without regard to a specific category.

\PmacSetup – True maintenance of a deployable PMAC application requires Pewin32. This collection of VIs is the start of set of VIs to download and maintain the configuration of PMAC's numerous P, Q, M, and I variables. For the purpose of speed, the VIs are implemented using CINs.

\PmacUtility - This collection provide controls and VIs to modify file paths, handle radio buttons, etc.

Documentation

PMACPanel has an extensive suite of documentation and help that can answer most of your questions. This manual along with the various PMAC User and Reference manuals contains thousands of pages of information on every aspect of PMAC and PMACPanel. There a numerous help files available and more being written all the time. Check the Delta Tau web site for documents and help files for your system.

\PmacDocument - contains electronic copies of this document, miscellaneous help items, and a few useful bitmaps.

Chapter 4 - Application Basics

Basics

This chapter contains several systematic exercises that guide you through the hierarchy of PMACPanel to introduce the various concepts required to develop your own applications. For detailed explanations of the individual VIs used in each tutorial, consult the VI Reference. If you have not used PMACPanel before this chapter must be read.

Although most PMACPanel VIs can be used as is, the developer is encouraged to use them as templates that can be extended and customized to meet the requirements of your own application. PMACPanel required over 1000 hours of development time – time that you don't have to expend to achieve rapid and surprising results.

LabVIEW Techniques for PMACPanel

The following are general LabVIEW programming techniques not related to a specific PMACPanel VI and therefore may not appear elsewhere in this manual. PMACPanel assumes that you have had a basic course in the use of LabVIEW or equivalent experience.

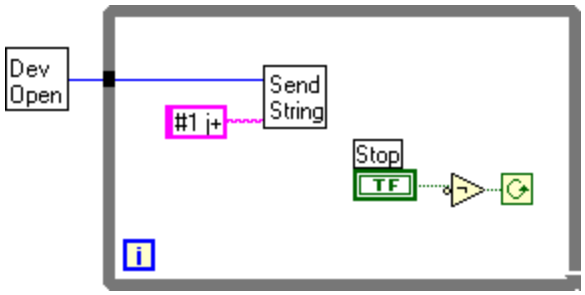
PMACPanel's architecture was designed to hierarchically encapsulate common operations into VIs that you can use to develop your own applications without doing a lot of basic communication parsing and wiring. Depending on your application's requirements and your experience with LabVIEW, you will have your own design patterns. The tutorial exercises that follow and the examples contained in the release reflect different ways to architect your applications to maximize the utility of PMACPanel. These techniques are used throughout PMACPanel and can be applied in your own application. For an excellent reference on LabVIEW techniques and application design issues see LabVIEW Graphical Programming by Gary Johnson (McGraw Hill ISBN 0-07-032915-X).

Dataflow and Sequencing

In LabVIEW the order of VI, execution is not guaranteed. Some PMAC operations require sequenced command execution. For example, a command to start a motion program should be sent before you go waiting for it to complete! In other situations, M-Variables must be defined before they can be used. Some PMACPanel VIs anticipate the need for sequencing and provide an output to enable this without using a LabVIEW sequence structure. In some instances, you must use a sequence structure.

Dataflow and Recurring Execution

LabVIEW loves to use while loops to execute VIs - again, and again, and again. If you want to continually send a "Jog+" command to PMAC, use the following example. It will send the command again, and again. This wastes PMAC's time and slows your PMACPanel application. To prevent this you, as the developer, must develop your program logic so that the commands are not repeatedly sent to PMAC. Most of PMACPanel's architecture is designed to simplify this for you by encapsulating this logic at the lowest levels possible.



Giving Up Control

Make sure that your PMAC VI's using concurrent execution loops use a wait timer to give the user interface and other VIs a fair shot at executing. Otherwise, you just may lock up the user interface while your PMACPanel program is waiting for a motion program to finish. Worse yet, other VI's that use double buffering to acquire data can overflow with nasty results

Execution Speed

LabVIEW is pretty fast at doing certain types of things and a little slower at others. Writing complex VI's such as the terminal is tough on LabVIEW execution and the developer. Don't expect the response time of visual C++ or Pewin32.

You should realize that applications that repeatedly poll PMAC can slow your application. This is especially true of systems that use serial communication at low baud rates. On the other extreme are busy applications that use the DPR capabilities of PMACPanel for high-speed data transfers.

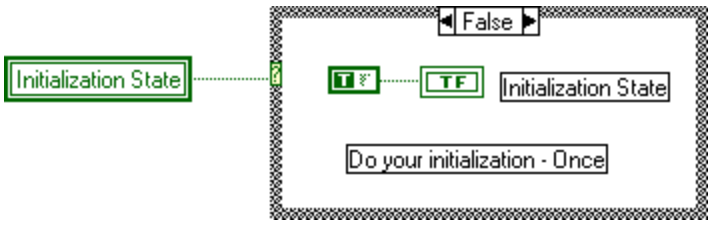
VI Reentrancy

In general, most VIs are reentrant. In LabVIEW terminology, this means that a VI can be used simultaneously in multiple PMACPanel and application VIs with its own separate copy of data. This has some benefits and some drawbacks. It allows independent simultaneous execution of the reentrant VI. The drawback is that it prevents each use of the VI from having a user accessible copy of the panel. In the case of low level PMACPanel VIs this is not really an issue because the VIs have no user interface value. It becomes more of an issue with the ICV VIs.

Issues can also arise when sending commands to PMAC. If multiple VIs are busy sending commands it is not only possible but also probable that expected responses will not line up with the commands. We will cover this later in the chapter.

Persistent VI State

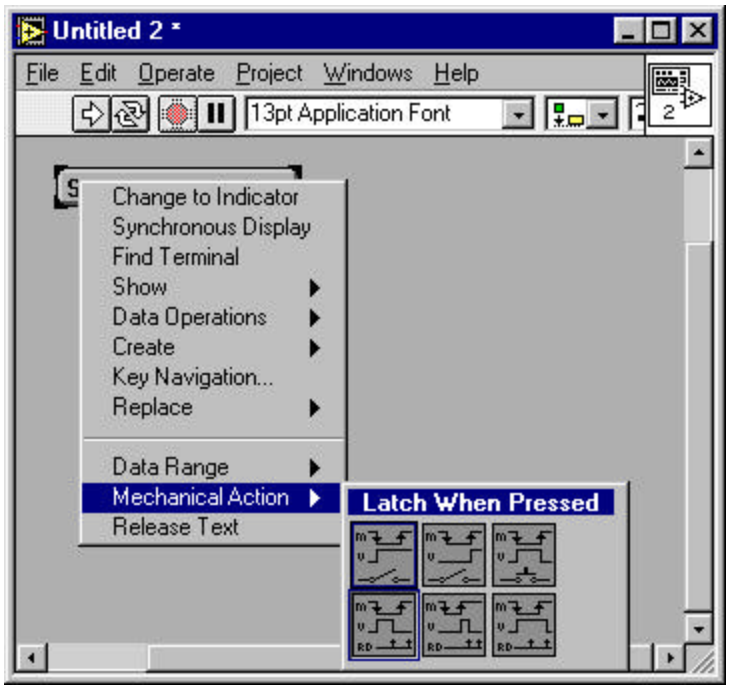
LabVIEW VIs maintain their state from execution to execution within a loop as long as they are loaded in memory. Sometimes this is desirable, and in many instances is used by PMACPanel VIs. Particularly those that are used to build configuration tables or attempt to minimize repetitive queries for data that will not change often. The following diagram demonstrates the use of this technique.



In this example the default value for Initialization State is FALSE. It is set to this value every time the VI is loaded into memory. When executed the first time, Initialization State is set to TRUE and will remain that way until it is reset by an operation in this VI or the calling VI is closed thereby unloading this from memory. Be aware that LabVIEW 5.0 has a bug handling this. National Instruments plans to release a fix for this problem with 5.0.1. Until then, PMACPanel 1.0 has added a few wires and local variables to fix this.

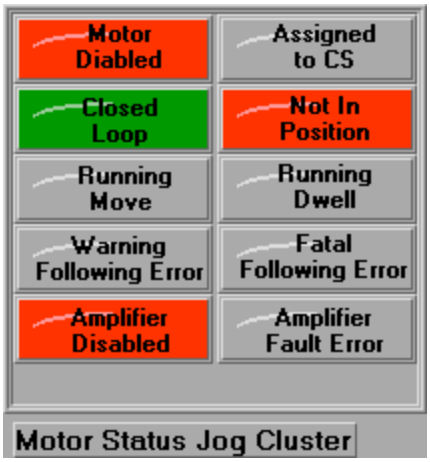
Mechanical Action

Most PMACPanel VIs requiring Boolean inputs or buttons have their mechanical action set to "Latch When Pressed". You can configure your buttons as shown in the figure below. Using this configuration, the button is read once when pressed and reset to FALSE. This is useful for preventing commands from being repeatedly sent to PMAC.



PMACPanel Indicator and Control Clusters

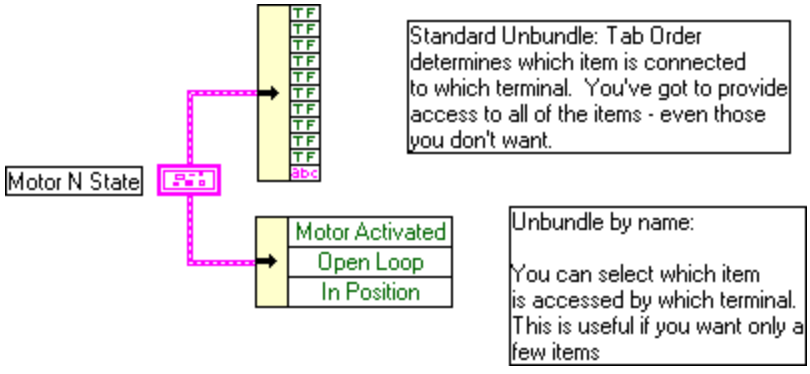
PMACPanel makes extensive use of predefined indicator and control clusters to make your development task easier. These clusters are easy to drop into your application. You should understand what clusters are, how to edit them, and how to access the individual controls and indicators they contain. An example of a Motor Status Jog Cluster **PmacMotorStatJog** is shown here.



A lot of work went into developing indicator clusters with proper names and item descriptions. You will find that when you have **Help»Show Help** enabled extensive descriptions of PMACPanel cluster items can help you understand what is displayed and what operations are performed by controls and indicators.

Cluster Item Access

The individual items of each PMACPanel cluster are named and given a Tab Order. Within PMACPanel, they are generally unbundled without the name so that the diagrams are a little easier to fit on a single page. The diagram below demonstrates the two different techniques for accessing the individual items of a cluster.

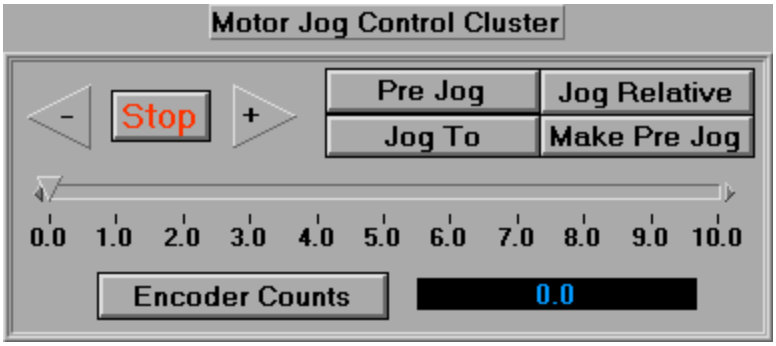

Motor N State
Motor Activated
Open Loop
In Position
Standard Unbundle: Tab Order determines which item is connected to which terminal. You've got to provide access to all of the items - even those you don't want.
Unbundle by name:
You can select which item is accessed by which terminal. This is useful if you want only a few items

Using the mouse on function VI terminals, you can easily create cluster constants if you require them. These can then be filled with the appropriate data.

Clusters Contain Controls or Indicators but not Both

Clusters are wonderful for grouping commonly used items together to make your life easier. A major limitation should be understood. In general, a cluster should not contain both indicators and controls. This doesn't work well with LabVIEW's data flow execution model - what happens if it sets an indicator item before it reads it as a control value? This is especially true of PMACPanel clusters that use Booleans configured with latched mechanical action. LabVIEW will not let this possible race condition go and generate an error.

PMACPanel has made some concessions for this. The **PmacMotorJogControl** control cluster shown below can be used to jog a motor. One would love to have a motor position indicator in the cluster! Sorry, but we created a separate position cluster.

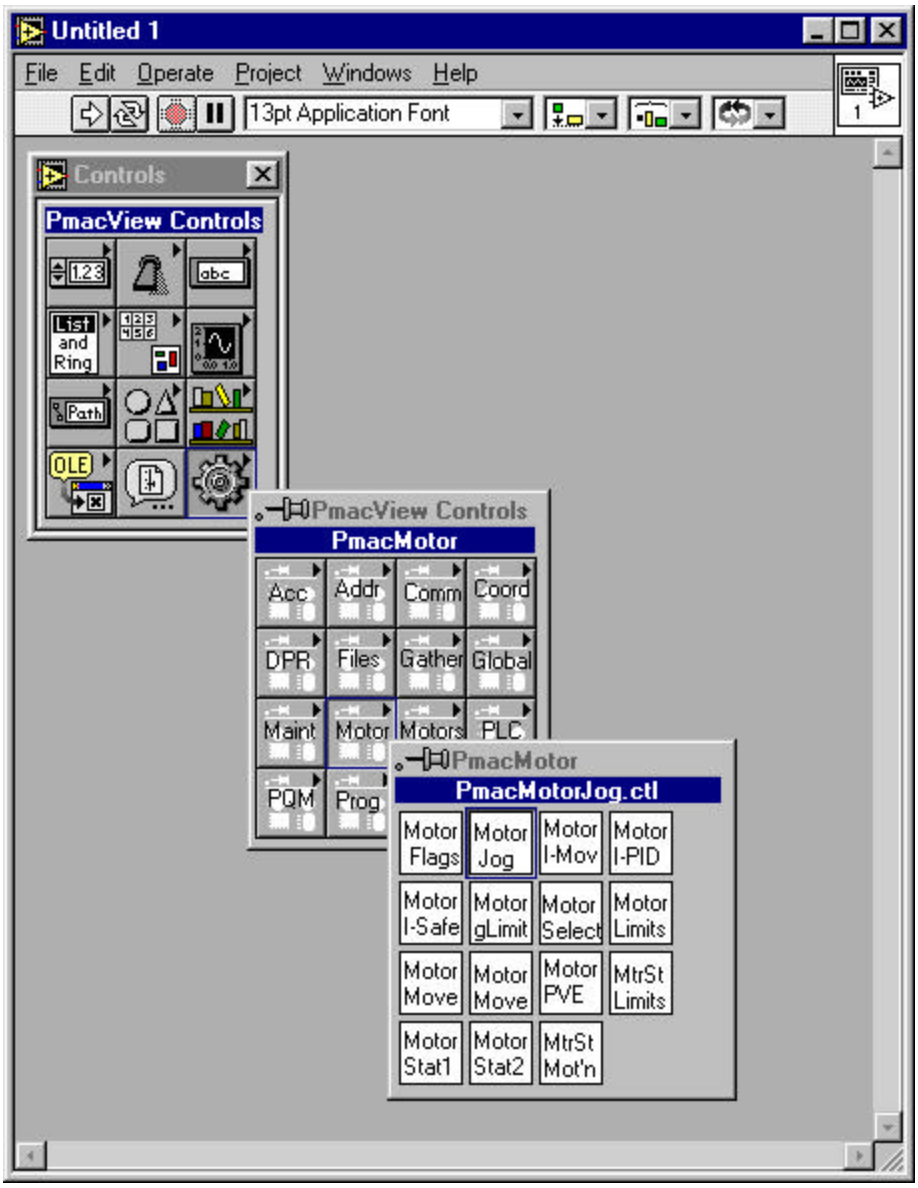


Cluster Type Definitions

LabVIEW allows you to define cluster controls and indicators as strict types that are linked to the root control definition. PMACPanel has chosen not to use this capability. Clusters can be used and modified independent of the base definition. This means that if you change cluster definitions the changes are not propagated to the VIs using it. You must replace all instances of the control by hand. You can always choose to define the *.ctl controls as strict types before you use them. Then every new instance of the control in your application will be linked to the raw control.

Accessing PMACPanel VIs

To use PMACPanel VIs select the PMACPanel control sub-palette attached to the PMACPanel.lib directory. Depending on the view you installed something like the following will appear.



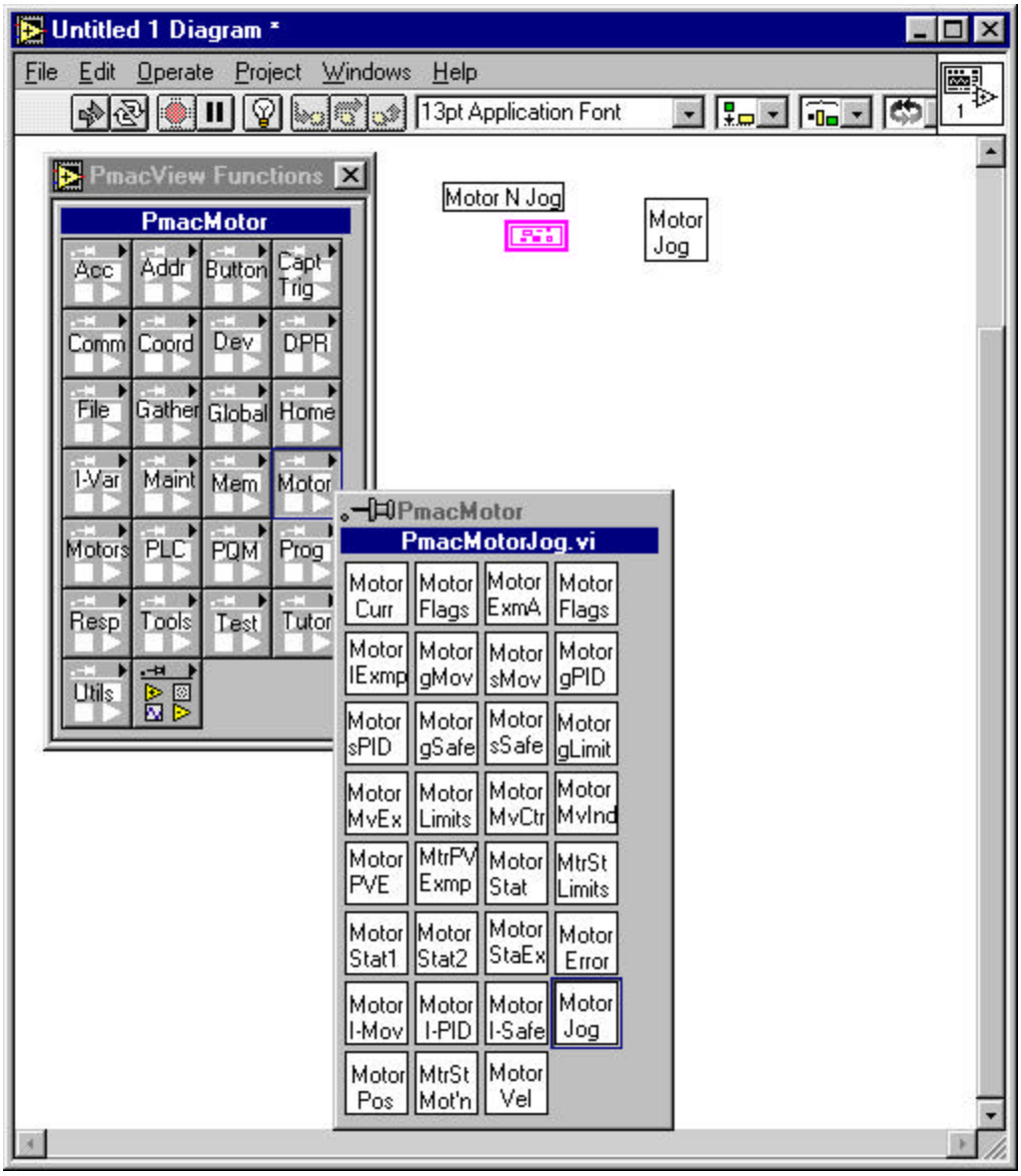
PMACPanel sub-palettes uses words rather than graphical icons to define their functionality just as their naming in the directory structure does. To place a PMACPanel Control on your panel the icon is selected from the palette and placed on your VIs panel.

Clusters With an Associated Function VI

PMACPanel controls exist on VI Panels. To get the data for indicators or generate commands from controls they need a PMACPanel function VI. To make it easier to link the two together the name of the associated function VI is the same as that of the control. We will say more about using these in the next section.

The figure shown below shows the terminal for **PmacMotorJogControl.ctl** and the function VI **PmacMotorJogControl.vi**. The similar names indicate that

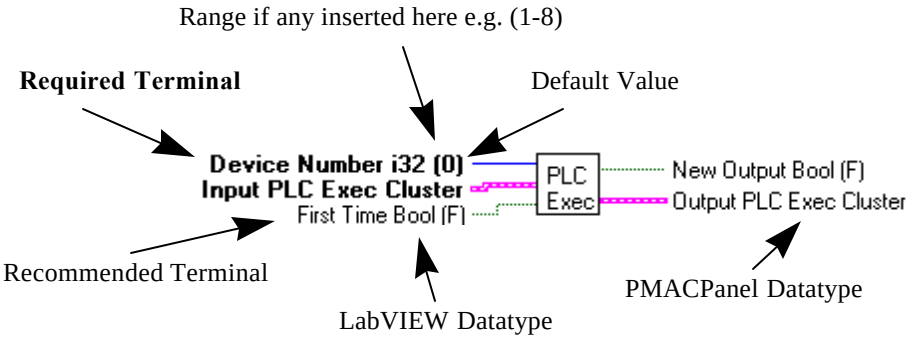
they are paired together and the panel cluster is wired somewhere on the function VI icon.



PMACPanel VI Terminal Conventions

PMACPanel has carefully chosen terminal names that are used consistently throughout the library. Extensive terminal descriptions and their behavior are available with LabVIEW's **Help»Show Help** facility and in the VI Reference chapter of the manual. To help you utilize the on-line help associated with PMACPanel VIs there are terminal naming conventions to identify the name, type, default value, and range of inputs and outputs. Most input terminals have a default value. Those that have a range coerce inputs to the range. Most inputs are required. A few are recommended. Their default action is fully defined.

Cluster inputs and outputs are typically named for the wire type expected. The following VI icon demonstrates these standards.



PMACPanel Tutorials

If you understand the basic ideas covered in the previous section you can start developing PMACPanel applications. The remainder of this chapter contains exercises that start with opening communication with PMAC and work up to the development of some very sophisticated capabilities. The contents of these tutorials are necessary for all PMACPanel developers. The on-line Windows Help version of this document is a great way to display the contents of the tutorials and examples from within LabVIEW.

Tutorial VIs are located in the **\PmacTutor** directory. In this directory there are two base VIs that are used as a starting place for developing all the examples and PMACPanel VIs. These are

- **PmacTutorApp** - Opens PMAC, has an execution while loop, and a Stop button
- **PmacTutorSub** - Has a predefined Device Number terminal

The tutorial VIs are named **PmacTutor1**, **PmacTutor2**, etc. These should be opened, examined, and executed while working through the exercises. You can save new copies of these and modify them as desired.

Depending on the exercise goals, various panels, diagrams, and sub-VI descriptions are used to illustrate PMACPanel concepts. Numerous descriptions are provided on the panels and in the diagrams.

Good wiring!

PMACPanel Communication Tutorial

The following exercises introduce those VIs used to send commands to PMAC and access its data. This includes special collections of VIs for:

- PComm32 access
- Sending commands

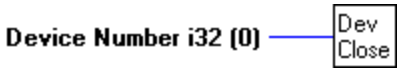
- Querying PMAC and processing responses
- Accessing PMAC memory-mapped data and variables
- Querying and Setting I-Variables

PmacTutor1- Accessing PComm32

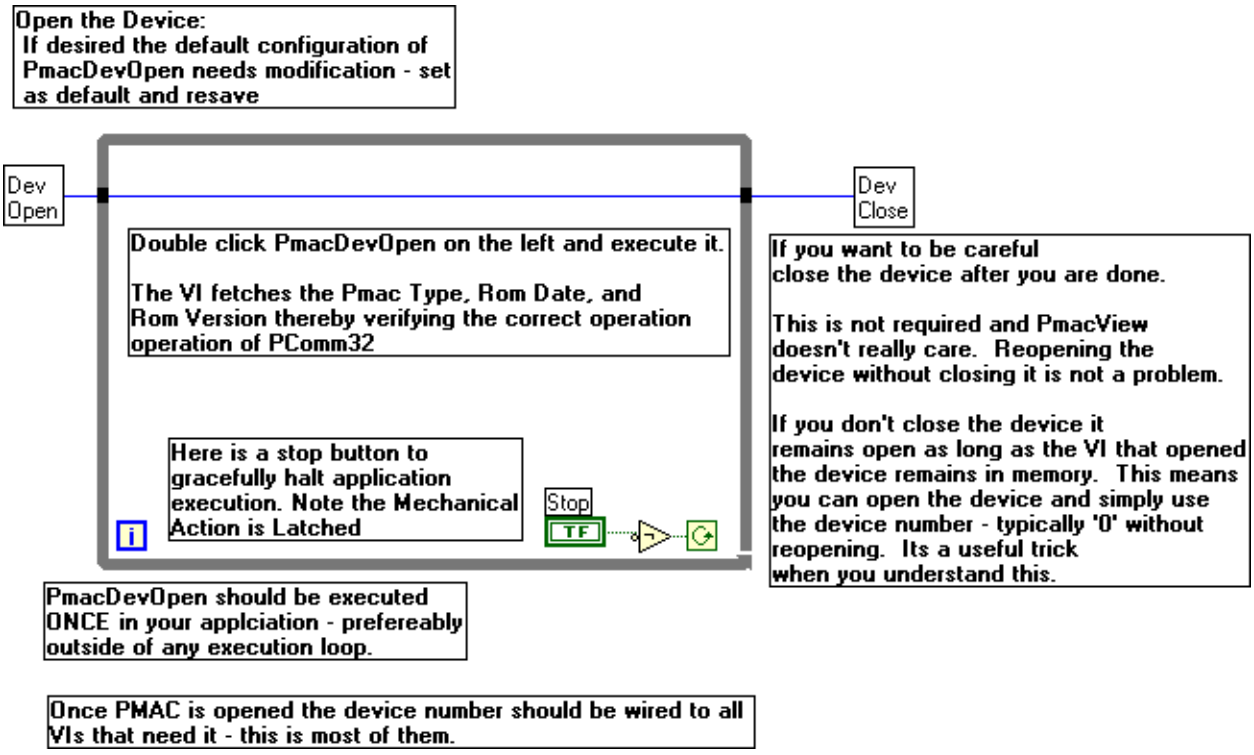
PmacTutor1 covers the basic requirements for accessing PComm32. All PMACPanel applications must open access to PComm32 using **PmacDevOpen**. This exercise demonstrates three steps for all PMACPanel applications. Open the device, do something, close the device. These VIs are

- **PmacDevOpen** - Open communication to PMAC using the PComm32 device driver. Check type, ROM date, and ROM Version. Provide Device Number for other VI's. You can select the mode of communication using the Communication Mode drop down menu. To make the selection permanent make your selection the default use the right mouse button and **Data Operations»Make Current Value Default** option. This MUST be done in conjunction with the options available on the PMAC control panel.

- **PmacDevClose** - Close the PComm32 device driver. PMAC will continue running as programmed as long as power is applied.



All basic PMACPanel application diagram will open PMAC and pass the device number into a your application's main loop where your primary logic is executed. Structuring the main loop this way establishes a dependency between the opening of the device and the execution of the rest of your application.


Open the Device:
If desired the default configuration of PmacDevOpen needs modification - set as default and resave
Dev Open
Dev Close
Double click PmacDevOpen on the left and execute it.
The VI fetches the Pmac Type, Rom Date, and Rom Version thereby verifying the correct operation operation of PComm32
If you want to be careful close the device after you are done.
This is not required and PmacView doesn't really care. Reopening the device without closing it is not a problem.
If you don't close the device it remains open as long as the VI that opened the device remains in memory. This means you can open the device and simply use the device number - typically '0' without reopening. Its a useful trick when you understand this.
Here is a stop button to gracefully halt application execution. Note the Mechanical Action is Latched
Stop
TF
PmacDevOpen should be executed ONCE in your applciation - prefereably outside of any execution loop.
Once PMAC is opened the device number should be wired to all VIs that need it - this is most of them.

Several techniques to ease the development process noted in the diagram descriptions. These should not be relied upon in your final application.

If you have more than one PMAC or wish to access your PMAC in more than one way it is easy to do.

PMACPanel was designed to be easy to use. To avoid having to provide **PmacDevOpen** with a Device Number, Communication Mode, etc. every time it is used the VI was configured with defaults for the Device Number and Communication Mode in Chapter 3. If you have more than one PMAC or wish to access the device using more than one mode you can use one of two equally simple methods.

- Make copies of **PmacDevOpen** and rename the copies something like **PmacDevOpenSerial** or **PmacDev0Open** and **PmacDev1Open**. Set the defaults for the Device Number and Communication Mode as desired. Use these exactly as you would **PmacDevOpen**.

- Make a copy of **PmacDevOpen** and add input terminals for Device Number and/or Communication Mode. When using this VI provide the inputs and use the outputs exactly as you would **PmacDevOpen**.

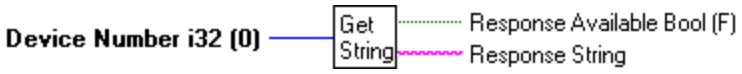
Multi-threading and PmacDevOpen

PMACPanel and PComm32 readily make use of LabVIEW's multi-threaded programming model. You can have multiple PMACPanel application VIs open PMAC and execute simultaneously without problems.

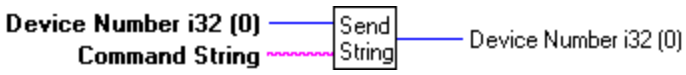
PmacTutor2 - Sending Commands and Getting Responses

The most basic interaction with PMAC is done using one of three **PmacComm** VIs

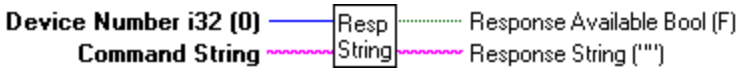
- **PmacCommGetStr** - Check if PMAC has data available. When Response Available is TRUE Response String contains all available data. When Response Available is FALSE Response String is the empty string. Responses are parsed for PMAC ERR codes and flagged with a modal dialog.



- **PmacCommSendStr** - Send Command String to PMAC. If Command String is the empty string, nothing is sent. The output Device Number is a copy of input Device Number to allow sequencing of commands to PMAC.



- **PmacCommRespStr** - Send Command String to PMAC and wait for a response. If Command String is the empty string, nothing is sent. Response Available is TRUE when Response String contains response data. When Response Available is FALSE Response String is the empty string. Responses are parsed for PMAC ERR codes and flagged with a modal dialog.



All the PMAC on-line commands described in the *PMAC User Manual* and *PMAC Software Reference Manual* are valid commands. See the appropriate manual for detailed command usage and syntax. PMAC will accept multiple commands in a command string. Those commands that generate a response will put the data into PMAC's output buffer whether or not you retrieve it. If a command generates a response, you should use **PmacCommRespStr**.

PMAC responses are generally single lines. The exceptions to this are commands like

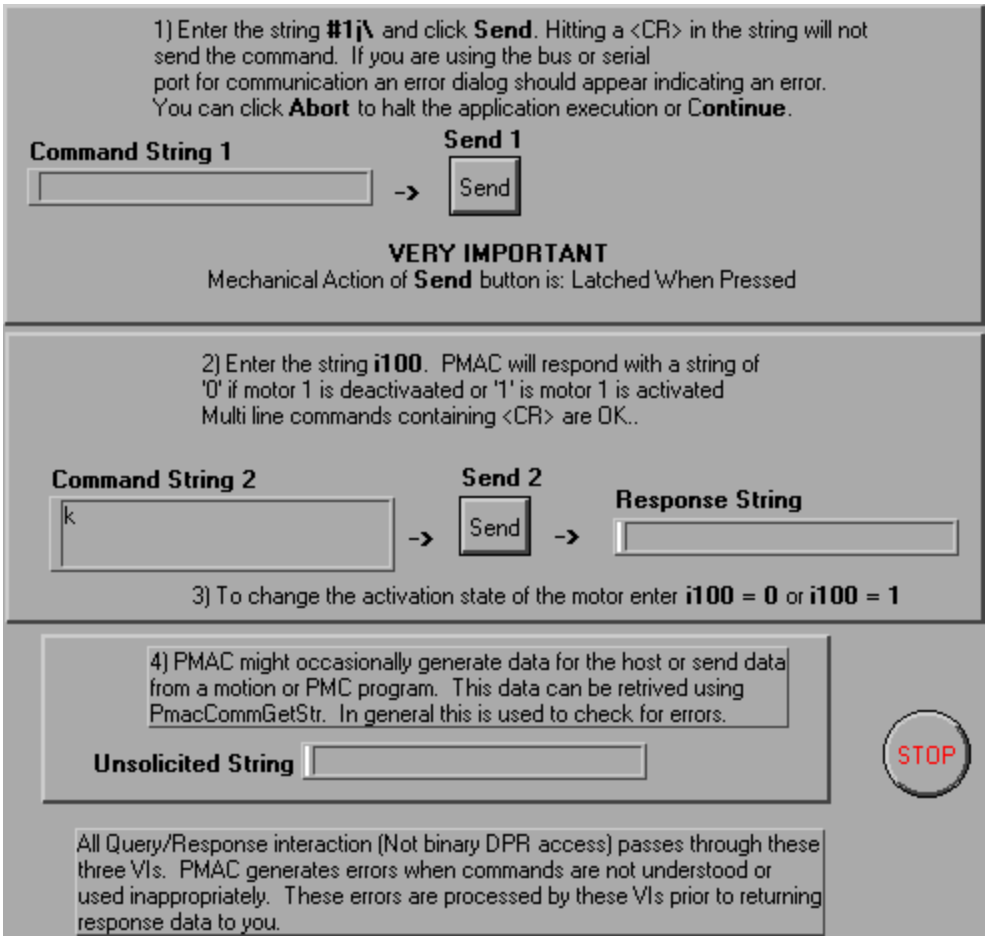
```
LIST PROG n
LIST GATHER
```

which will generate long multi-line responses. LabVIEW strings are happy to handle these. Depending on the size of the string indicator or control on your panel these may not wrap correctly. This is a LabVIEW issue. If you execute **PmacTerminal** and list a long gather buffer you will see this. There is also an issue when entering strings using a control. The natural temptation is to expect that hitting <CR> will cause the string to be sent. LabVIEW doesn't work this way. Panel values are input using the <Enter> key.

LabVIEW string controls do not treat a <CR> as anything other than a <CR>. It is possible to tie this keystroke to a control on the panel. Check the LabVIEW documentation on

Key Navigation Option for Controls

The panel and diagram for this exercise demonstrate the use of these three communication VIs. Execute the operations specified in step 1 through 4 on the panel.


1) Enter the string #1j\ and click Send. Hitting a <CR> in the string will not send the command. If you are using the bus or serial port for communication an error dialog should appear indicating an error. You can click Abort to halt the application execution or Continue.
Command String 1
Send 1
Send
VERY IMPORTANT
Mechanical Action of Send button is: Latched When Pressed
2) Enter the string i100. PMAC will respond with a string of '0' if motor 1 is deactivaated or '1' is motor 1 is activated Multi line commands containing <CR> are OK..
Command String 2
k
Send 2
Send
Response String
3) To change the activation state of the motor enter i100 = 0 or i100 = 1
4) PMAC might occasionally generate data for the host or send data from a motion or PMC program. This data can be retrived using PmacCommGetStr. In general this is used to check for errors.
Unsolicited String
STOP
All Query/Response interaction (Not binary DPR access) passes through these three VIs. PMAC generates errors when commands are not understood or used inappropriately. These errors are processed by these VIs prior to returning response data to you.

When you execute step 1 in this exercise, the following error dialog will appear informing you of a problem with your command. If the dialog does not appear, see the section PMAC Communication I-Variables in Chapter 2 to modify your PMAC's communication configuration as specified.

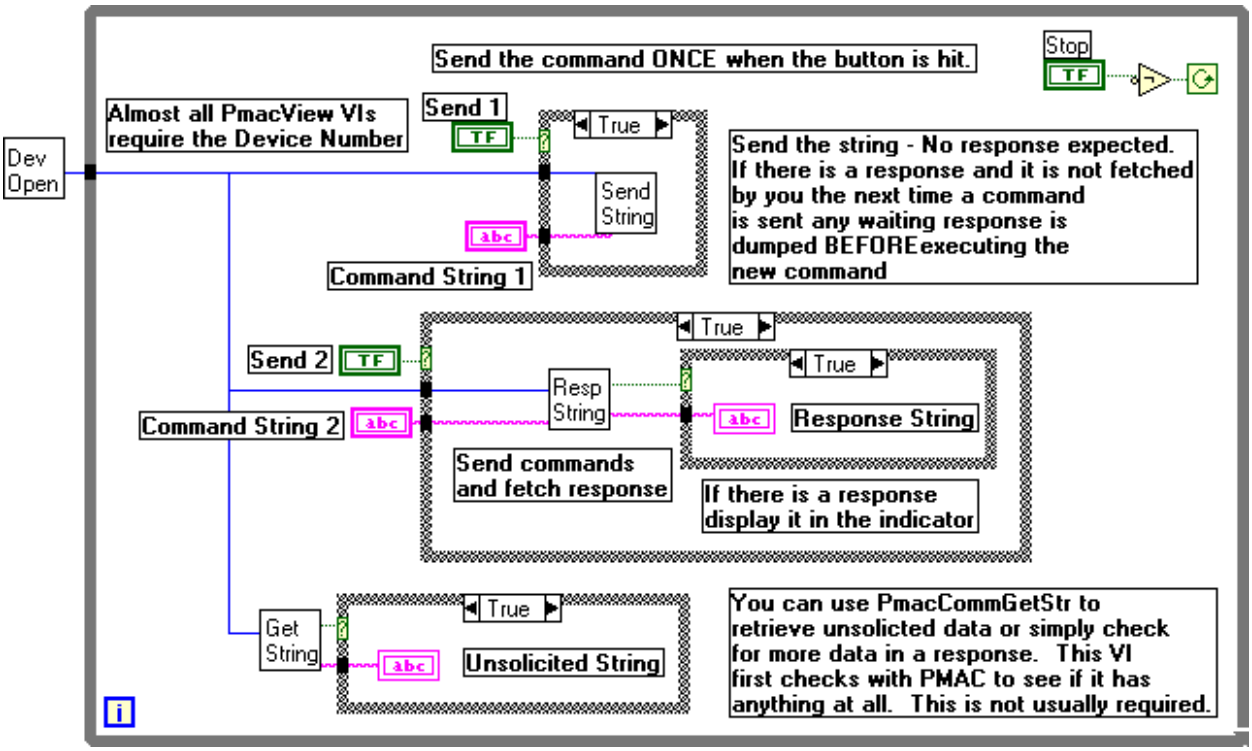
CAUTION *Until proper HW safeties have been installed, configured, and tested extreme caution must be exercised when moving motors to prevent damage and possible injury! Do not send a Jog command unless you are certain your actions will not damage your system or you!*

The error dialog appears because the command string "#1j\" is unrecognizable to PMAC. The correct syntax for the command is "#1j/". When error dialogs appear you have the choice of aborting your application or continuing. Commands that generate errors are not executed by PMAC and cause no harm. However, if your application logic continually attempts to send a bad command to PMAC you may have no choice but to abort the application. Otherwise, you may continue to get this dialog repeatedly. Your chances of halting the program using the standard LabVIEW STOP button before your application attempts to send the offending command again is unlikely. A complete listing of PMAC error codes can be found in the *PMAC User Manual* and *PMAC Software Reference Manual*.

The diagram for this tutorial demonstrates two basic things you must consider when designing you application.

If you send commands to PMAC in response to an interface button click the mechanical action of the button should be latched and the PMACPanel VI that sends the command should be in a Case structure. Otherwise, the command will be sent every iteration of the loop.

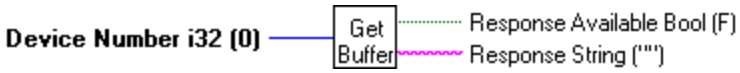
PmacCommRespStr and **PmacCommGetStr** indicate whether they received a response using Response Available. Response String will be the Empty String if nothing was received. It is generally good programming practice to test Response Available before using Response String.


Send the command ONCE when the button is hit.
Stop
Almost all PmacView VIs require the Device Number
Send 1
True
Dev Open
Send String
Send the string - No response expected. If there is a response and it is not fetched by you the next time a command is sent any waiting response is dumped BEFOREexecuting the new command
Command String 1
True
Send 2
True
Resp String
Command String 2
Response String
Send commands and fetch response
If there is a response display it in the indicator
Get String
True
Unsolicited String
You can use PmacCommGetStr to retrieve unsolicted data or simply check for more data in a response. This VI first checks with PMAC to see if it has anything at all. This is not usually required.

Using these three basic VIs you could generate an entire, albeit complex, PMACPanel application. The purpose of most of PMACPanel is to prevent you from having to do this!

PMAC and PComm32 limit basic responses to 256 characters. **PmacCommRespStr** and **PmacCommGetStr** handle this internally using the VI **PmacCommGetBuffer** to retrieve longer responses. Your applications will generally not make use of this VI.

- **PmacCommGetBuffer** - Check if PMAC has data available. When Response Available is TRUE Response String contains all available data. When Response Available is FALSE Response String is the empty string. Responses are not parsed for PMAC ERR codes.



Exercise **PmacTutor3** shows how PMACPanel simplifies the sending of commands in response to panel buttons and Boolean conditions.

PmacTutor2a - Communication Logging

This is a rather advanced topic, but one that we include here for completeness. PMACPanel has the ability to log all communication between your application and PMAC. You might use this to monitor what your users are doing or want to log interesting sessions for later play back. PMACPanel maintains all communication using the following VIs. Future versions of PMACPanel may use this capability to implement a Graphical Motion Language.

- **PmacCommGlobal** - This VI is a global copy of the PmacCommGlobal cluster used by several PmacComm VIs for error reporting and logging purposes.

Pmac Communication Cluster This cluster maintains a log of communication between PMACPanel and PMAC.

Command String Last on-line command sent to PMAC

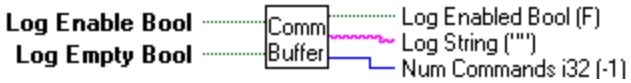
Response String Last response received from PMAC

Communication Log String A multi-line buffer of commands sent to PMAC and received from PMAC.

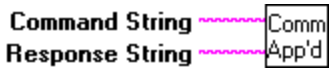
Num Commands i32 The number of commands sent to PMAC and logged in the Communication Buffer.

Buffer Log Bool (F) When TRUE all communication is appended and logged to the Communication Log.

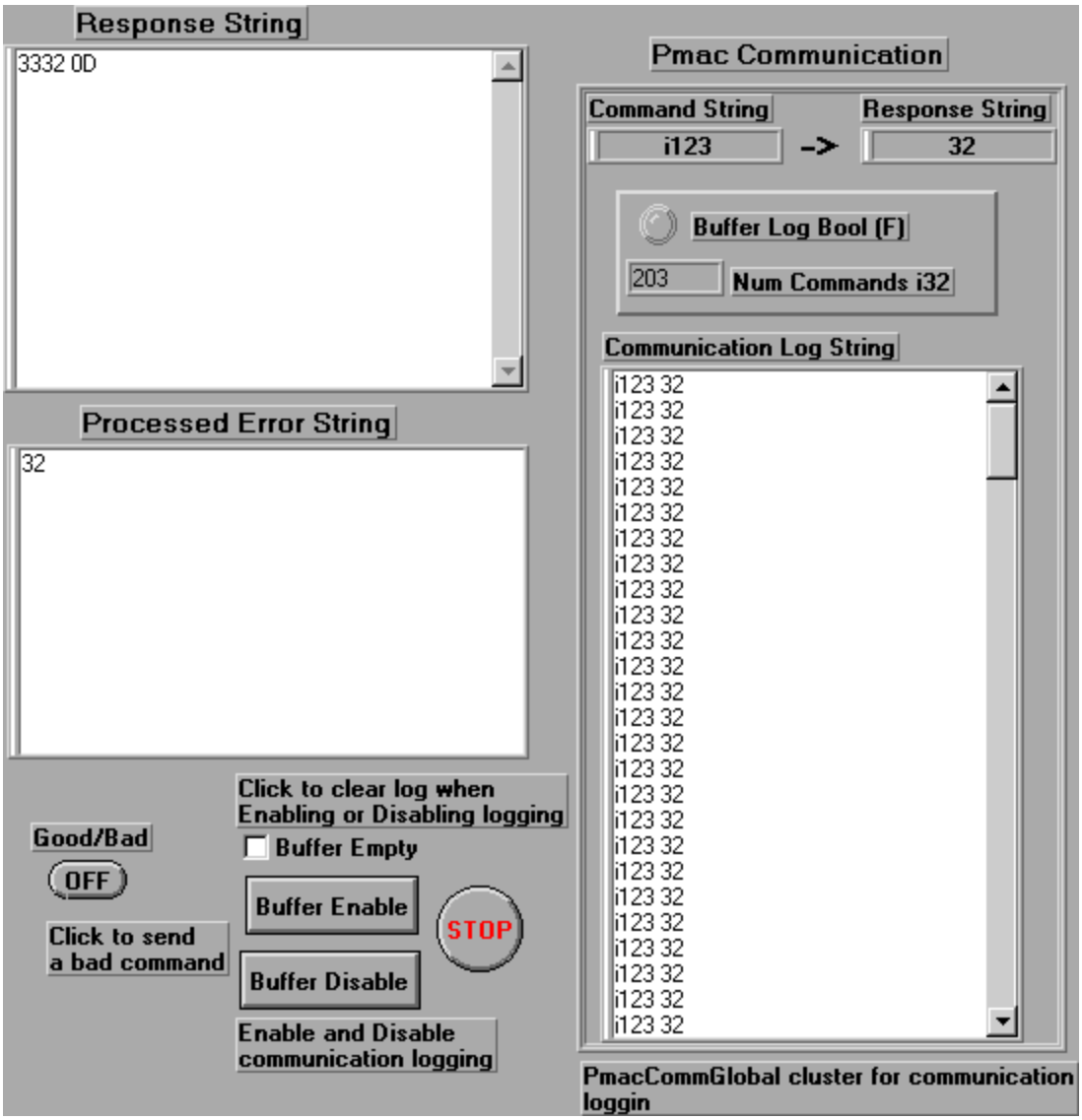
- **PmacCommBuffer** - When Log Enable is TRUE communication logging is enabled. Log Enabled Bool, Log String, and Num Commands reflect the state of the log buffer when logging is enabled. Log String is the empty string, Num Commands = -1, and Log Enabled is FALSE when logging is disabled. When Log Empty is TRUE the log buffer is emptied.



- **PmacCommAppend** - Copy Command String and Response String to the Last Communication items in PmacCommGlobal. If Logging Enabled is TRUE they are also appended to the Communication Log.

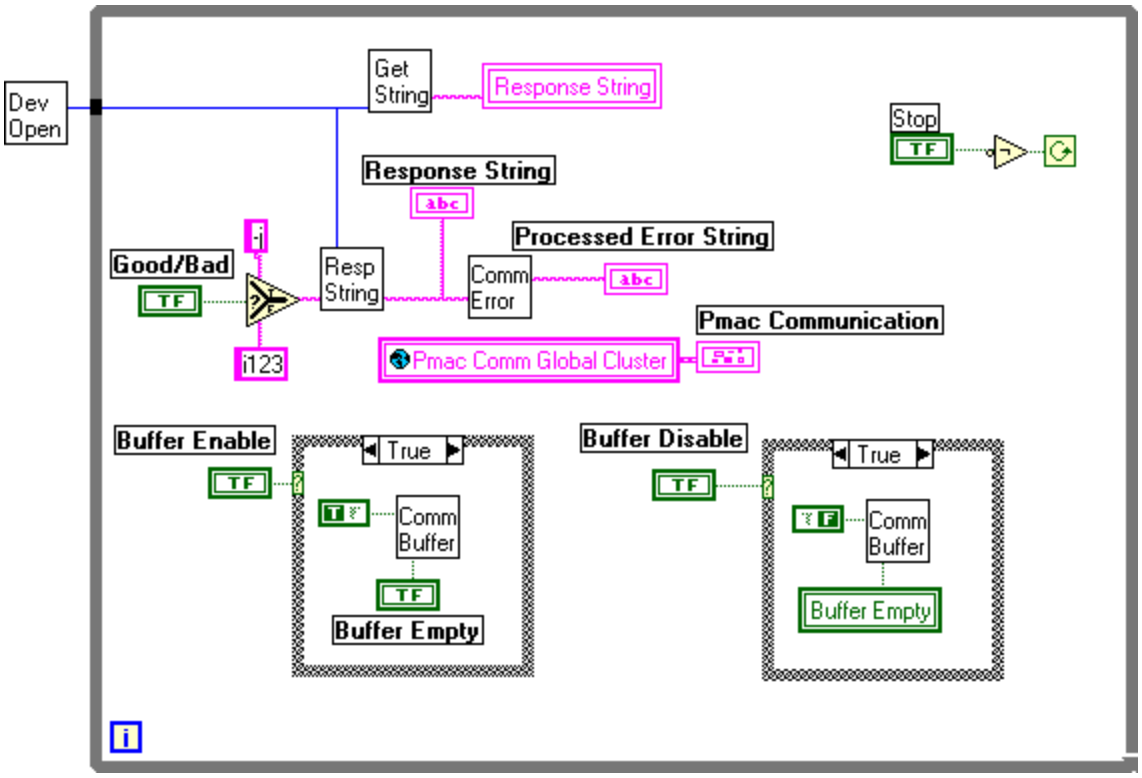


The panel for this exercise demonstrates how communication is logged. The VI queries PMAC for the value of i123. When Good/Bad is clicked the incorrect command -j is sent resulting in an error. You will see the log of this bad command in the Command String and Response String items in the PMACommunication Cluster on the right. You will also see the error dialog pop-up allowing you to continue or abort the application - click Continue.



If you click the button Buffer Enable all communication is appended to Communication Log Buffer in the cluster. You can stop logging by clicking Buffer Disable. This only stops the logging of communications. If you click the Buffer Empty box and click Buffer Enable, the buffer is cleared before logging is enabled.

The diagram for this exercise is shown here.



If you desire to use logging in your application you need to develop logic to save the Log Buffer to a file. You should also realize that the size of this buffer could grow VERY large if your application uses status-monitoring ICVs and you don't save and empty the contents of the log buffer at reasonable intervals.

PmacTutor3 - Sending Commands Using Buttons

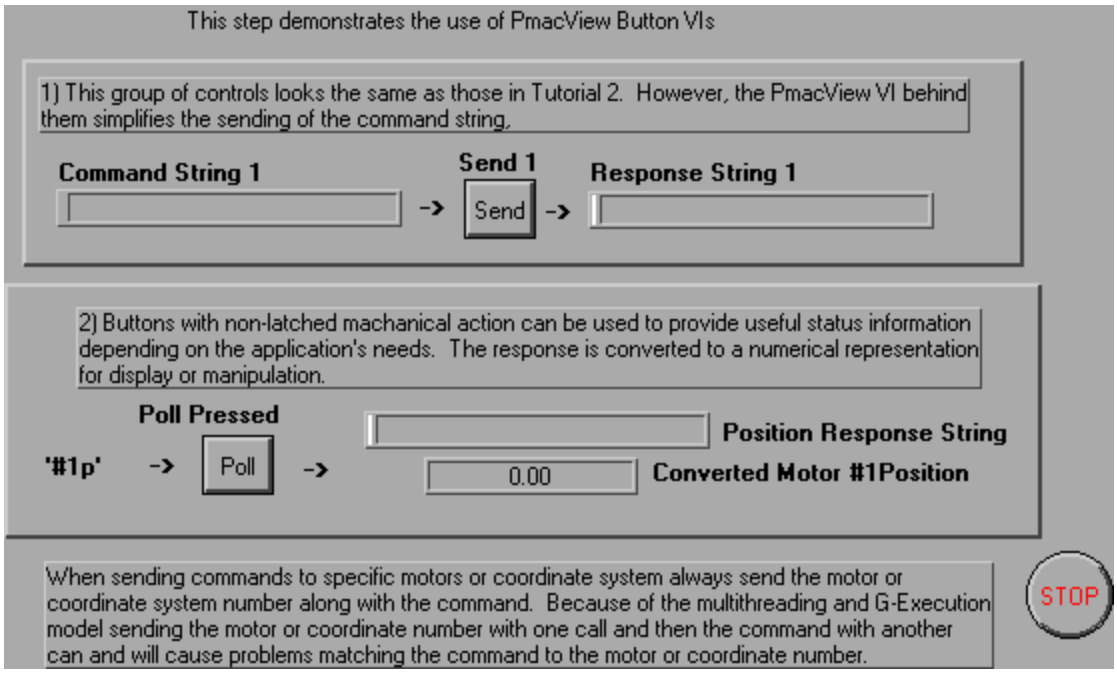
PMACPanel contains the **PmacButton** collection of VIs that send a command string to PMAC when an input button state is TRUE. Your applications panels will make constant use of these capabilities.

- **PmacButtGetStr** - Send Command String to PMAC and wait for a response when Button State is TRUE. When Response Available is TRUE Response String contains the response. If Response Available is FALSE Response String defaults to the empty string.

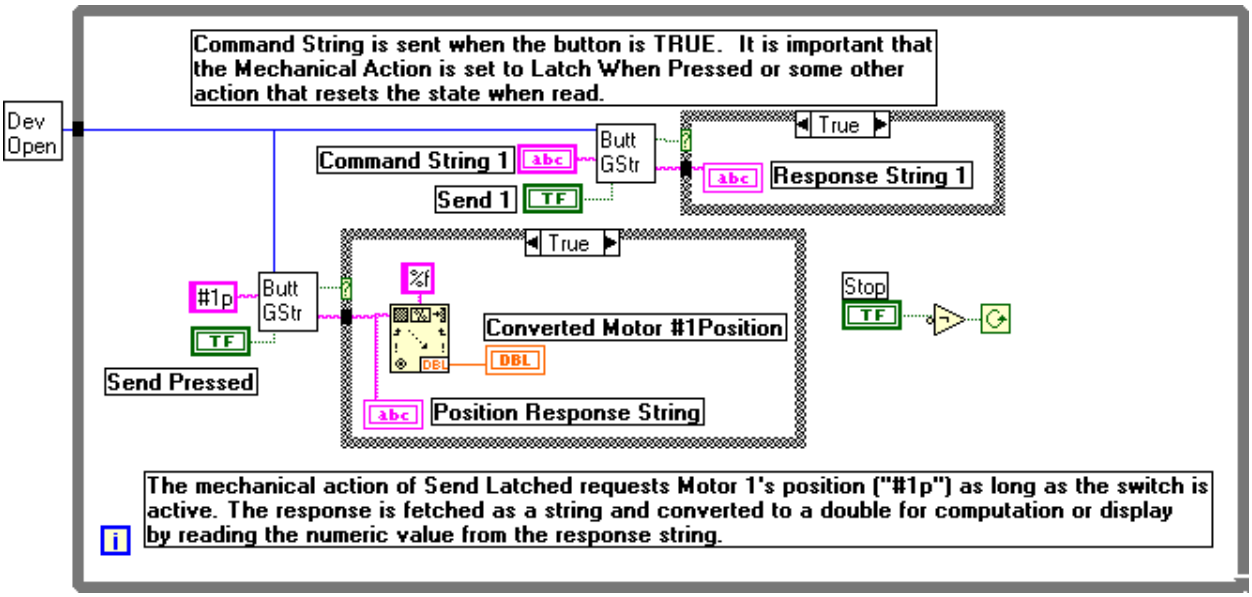
- **PmacButtSendStr** - Send Command String to PMAC when Button State is TRUE. Response Available is TRUE when PMAC has processed the command.

You should use **PmacButtGetStr** for commands that expect responses and **PmacButtSendStr** for commands that do not expect responses. **PmacButtSendStr** doesn't return a response so the input Button State is passed through to facilitate execution sequence dependencies.

The panel shown here demonstrates the use of these VIs, the conversion of PMAC responses into numeric data for use in LabVIEW, and the use of non-latched mechanical action to enable polled status for real-time.


This step demonstrates the use of PmacView Button VIs
1) This group of controls looks the same as those in Tutorial 2. However, the PmacView VI behind them simplifies the sending of the command string.
Command String 1
Send 1
Send
Response String 1
2) Buttons with non-latched machanical action can be used to provide useful status information depending on the application's needs. The response is converted to a numerical representation for display or manipulation.
Poll Pressed
'#1p'
Poll
Position Response String
0.00
Converted Motor #1Position
When sending commands to specific motors or coordinate system always send the motor or coordinate system number along with the command. Because of the multithreading and G-Execution model sending the motor or coordinate number with one call and then the command with another can and will cause problems matching the command to the motor or coordinate number.
STOP

You will note in the diagram for this VI that the sending of the command to PMAC is simplified by the use of the **PmacButtGetStr**. The button is directly supplied to the VI rather than wrapping the **PmacCommRespStr** VI in your own case structure. Sending commands in response to buttons is so common that this added capability makes application development significantly simpler.


Command String is sent when the button is TRUE. It is important that the Mechanical Action is set to Latch When Pressed or some other action that resets the state when read.
Dev Open
Command String 1
Butt GStr
Send 1
TF
True
Response String 1
#1p
Butt GStr
TF
Send Pressed
True
%f
Converted Motor #1Position
DBL
Position Response String
Stop
TF
The mechanical action of Send Latched requests Motor 1's position ("#1p") as long as the switch is active. The response is fetched as a string and converted to a double for computation or display by reading the numeric value from the response string.

The second step in the diagram sends the command "**#1p**" to PMAC requesting the position for motor #1 as long as the button is TRUE. The response string is displayed both as a string and converted into a numerical value for display. When sending any motor or coordinate system specific command to PMAC you should include the motor or coordinate system number with the command to prevent LabVIEW's execution order from sending addresses and commands in whatever order it desires.

The next exercise demonstrates how PMACPanel simplifies these operations further.

PmacTutor4 - Button and Response VIs

The **PmacResponse** VIs send commands to PMAC and convert the ASCII response string into LabVIEW numeric data types. This relieves you of having to scatter string conversion VIs all over your application. There are more members in the **PmacButton** collection introduced in PmacTutor3 that use the **PmacResponse** VIs to convert ASCII string responses into numeric data.

PmacResponse

See LabVIEW and PMAC Numeric Data Types in Chapter 3.

PmacResponse consists of 6 VIs - one for each type of numeric response conversion supported. If you have a data type you truly want to support, you can easily add it by copying and modifying one of these.

When using these VIs, refer to the *PMAC Software Reference Manual*; determine the size of the response, whether it will be signed or unsigned, and whether you will be manipulating the bits of the response. **PmacRespGetDbl** is used to introduce the collection.

- **PmacRespGetDbl** - If Command String is not the empty string send it to PMAC and wait for a response. If Response Available is TRUE Response contains a valid response. Otherwise, Response is 0.0.

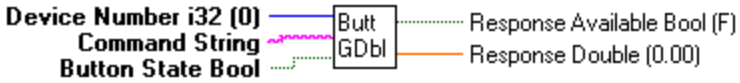
The remaining five VIs operate the same and simply provide responses of the appropriate type.

- **PmacRespGetBool**
- **PmacRespGetShort**
- **PmacRespGetUShort**
- **PmacRespGetLong**
- **PmacRespGetULong**

PmacButton

PmacButton consists of six additional VIs beyond **PmacButtGetStr and PmacButtSendStr** introduced in **PmacTutor3**. These six additional VIs provide numerical responses. **PmacButtGetDbl** is used to introduce the collection.

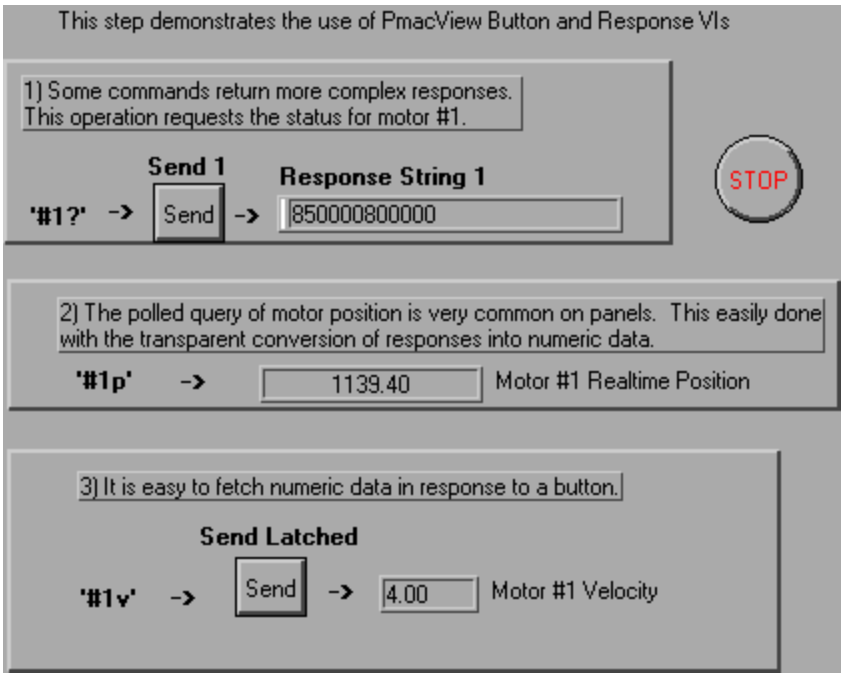
- **PmacButtGetDbl** - Send Command String to PMAC and wait for a response when Button State is TRUE. When Response Available is TRUE Response Double contains the response. If Response Available is FALSE Response Double defaults to 0.0.



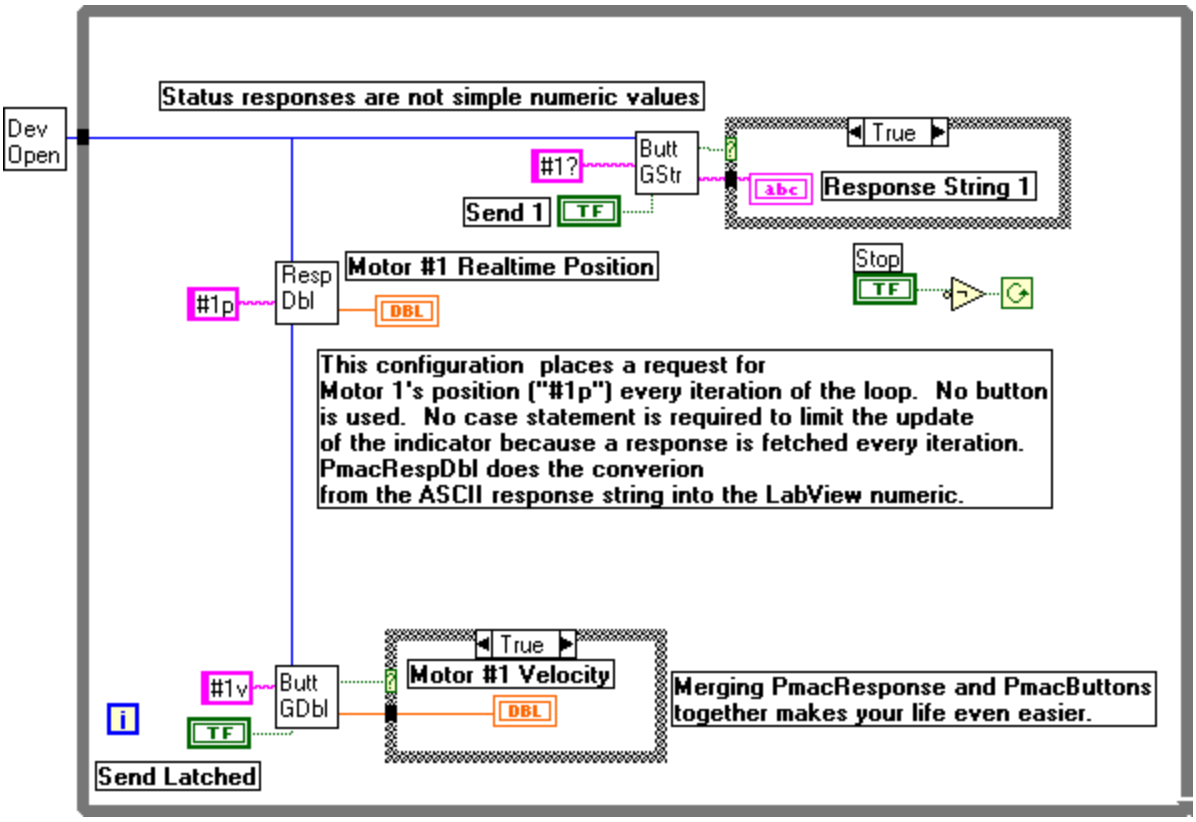
The remaining five VIs operate the same and simply provide responses of the appropriate type.

- **PmacButtGetBool**
- **PmacButtGetShort**
- **PmacButtGetUShort**
- **PmacButtGetLong**
- **PmacButtGetULong**

Step 1 of the exercise demonstrates that some response data is a little more than a numerical value. Step 2 demonstrates how a single **PmacResponse** VI can be used to provide a useful piece of data for your panel. When coupled with the button concept, PMAC data can be requested and converted using a single VI.


This step demonstrates the use of PmacView Button and Response VIs
1) Some commands return more complex responses.
This operation requests the status for motor #1.
Send 1
Response String 1
'#1?' ->
Send
->
850000800000
STOP
2) The polled query of motor position is very common on panels. This easily done with the transparent conversion of responses into numeric data.
'#1p' ->
1139.40
Motor #1 Realtime Position
3) It is easy to fetch numeric data in response to a button.
Send Latched
'#1v' ->
Send
->
4.00
Motor #1 Velocity

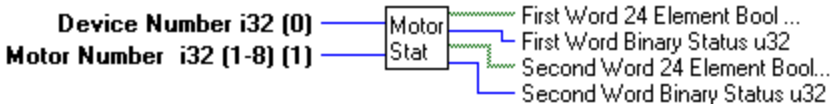
Although the operations are more complex, the diagram is simpler. As we progress through more exercises, the real power of PMACPanel will become apparent.



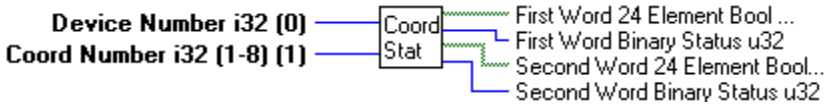
PmacTutor5 - Accessing PMAC Status

PmacTutor4 demonstrated the conversion of PMAC responses into numerical LabVIEW data types. Several responses require conversions that are more sophisticated. PMAC status is returned as 12 hexadecimal characters for a total of two 24-bit status words. Because status is critical to successful integration of PMAC with your LabVIEW application there are three VIs that request status and convert it into two unsigned integers and two 24 element Boolean arrays. The Boolean array representations allow you to select individual status bits for your own use using LabVIEW index VIs. The unsigned integers can be used for your own bit manipulation and testing using logical operators. These VIs are

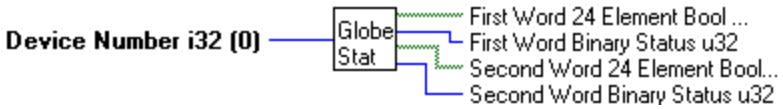
- **\PmacMotor\PmacMotorStat** - Query PMAC for the status of Motor Number. Report the two status words as arrays of Booleans and unsigned 32 bit integers.


Device Number i32 (0)
Motor Number i32 (1-8) (1)
Motor Stat
First Word 24 Element Bool ...
First Word Binary Status u32
Second Word 24 Element Bool...
Second Word Binary Status u32

- **\PmacCoord\PmacCoordStat** - Query PMAC for the status of the CS specified by Coord Number. Report the two status words as arrays of Booleans and unsigned 32 bit integers.


Device Number i32 (0)
Coord Number i32 (1-8) (1)
Coord Stat
First Word 24 Element Bool ...
First Word Binary Status u32
Second Word 24 Element Bool...
Second Word Binary Status u32

- **\PmacGlobal\PmacGlobalStat** - Query PMAC for PMAC's global status. Report the two status words as arrays of Booleans and unsigned 32 bit integers.

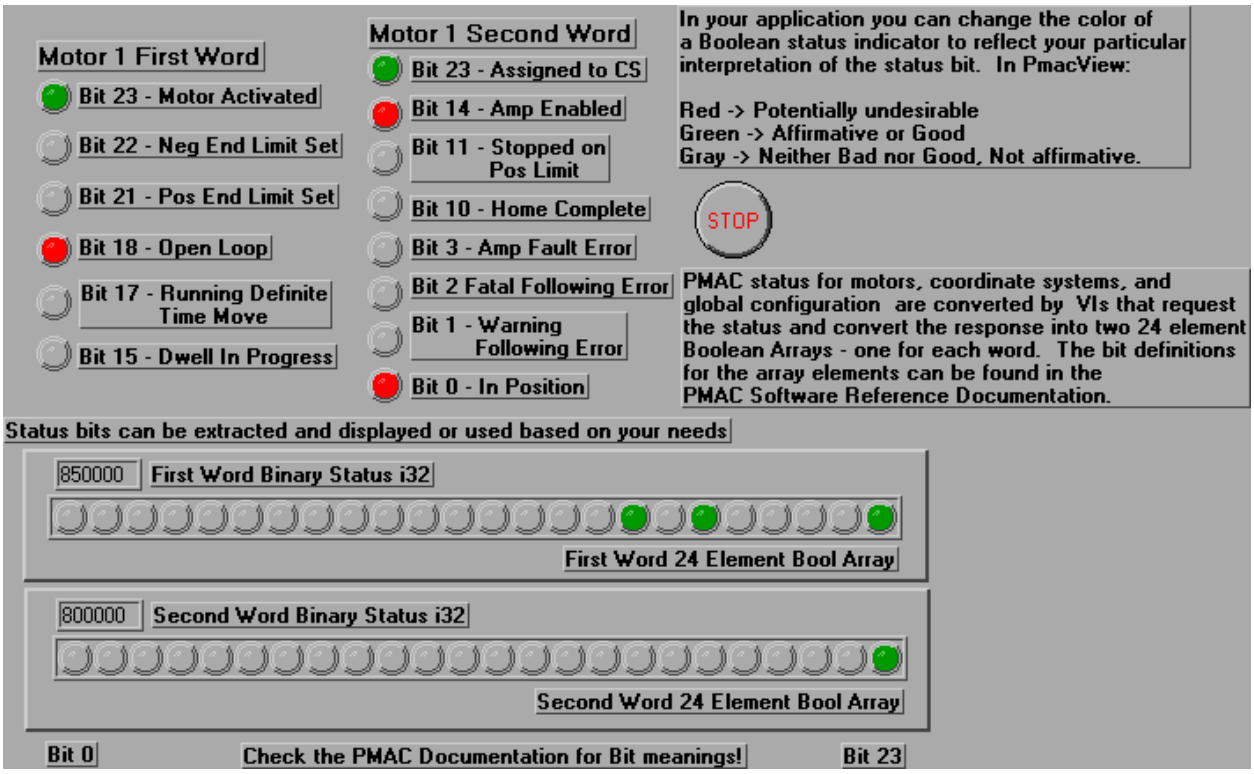

Device Number i32 (0)
Globe Stat
First Word 24 Element Bool ...
First Word Binary Status u32
Second Word 24 Element Bool...
Second Word Binary Status u32

PMAC supports up to 8 motors and coordinate systems. If your PMAC has fewer motors PMAC ignores commands to them.

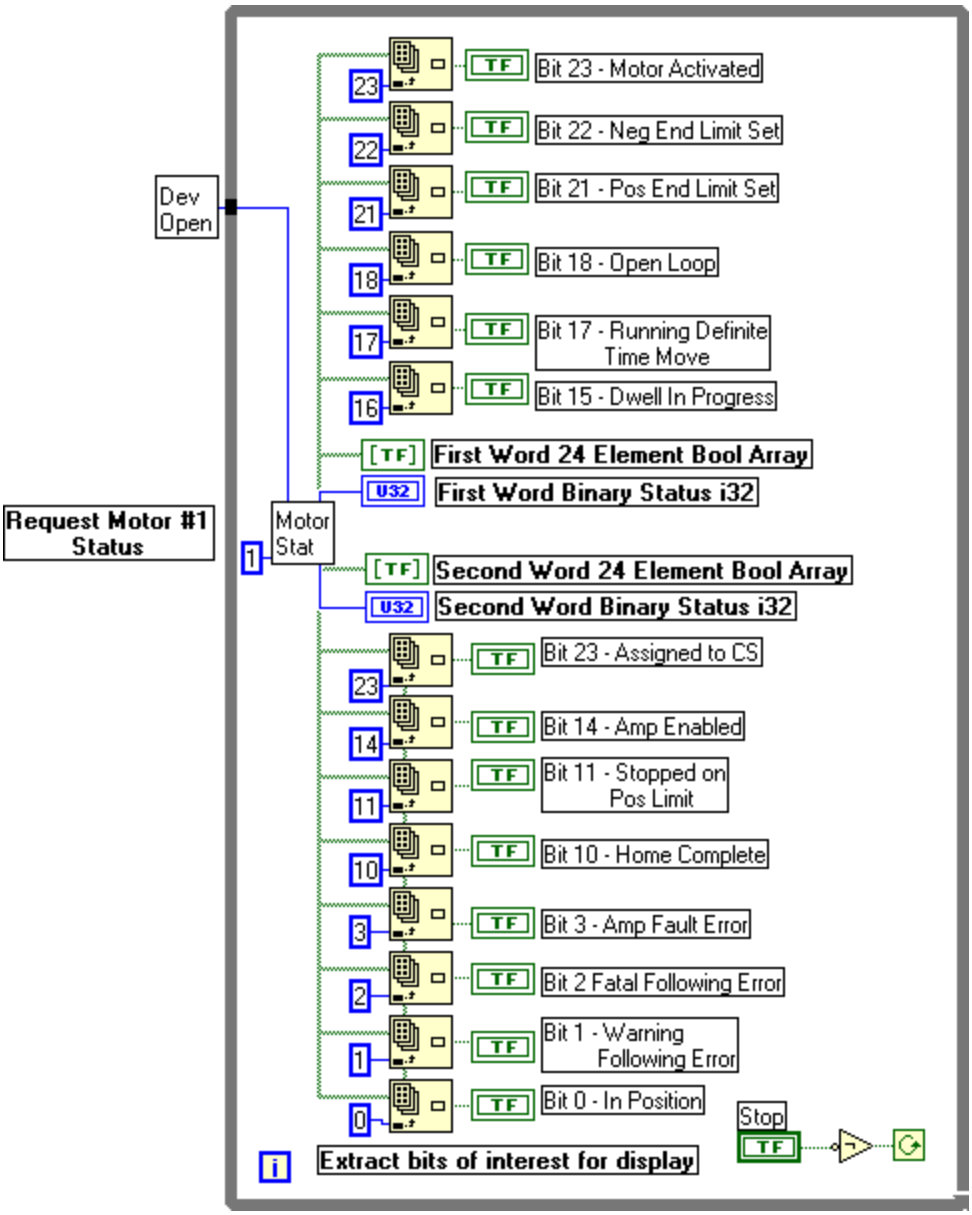
Global status does not require a motor number or coordinate system number. The Motor Number and Coord Number inputs coerce the range to protect you from mistakes. Status bit definitions can be found in the *PMAC Software Reference Manual*. More status processing VIs are introduced in later exercises.

PMACPanel does not supply button versions of these VIs that fetch status when the button input is TRUE. Status monitoring is generally not a user driven operation.

The panel for this exercise demonstrates the use of **PmacMotorStat**. Use of the coordinate system and global status VIs is identical. At the bottom of the panel are the raw status Boolean arrays. You cannot specify Boolean text for individual array elements hence they are left unlabeled. The contents of these arrays can be indexed using standard LabVIEW array function VIs to select specific bits for your needs. In this example, several common status bits are used to drive indicators.


Motor 1 First Word
Bit 23 - Motor Activated
Bit 22 - Neg End Limit Set
Bit 21 - Pos End Limit Set
Bit 18 - Open Loop
Bit 17 - Running Definite Time Move
Bit 15 - Dwell In Progress
Motor 1 Second Word
Bit 23 - Assigned to CS
Bit 14 - Amp Enabled
Bit 11 - Stopped on Pos Limit
Bit 10 - Home Complete
Bit 3 - Amp Fault Error
Bit 2 Fatal Following Error
Bit 1 - Warning Following Error
Bit 0 - In Position
In your application you can change the color of a Boolean status indicator to reflect your particular interpretation of the status bit. In PmacView:
Red -> Potentially undesirable
Green -> Affirmative or Good
Gray -> Neither Bad nor Good, Not affirmative.
STOP
PMAC status for motors, coordinate systems, and global configuration are converted by VIs that request the status and convert the response into two 24 element Boolean Arrays - one for each word. The bit definitions for the array elements can be found in the PMAC Software Reference Documentation.
Status bits can be extracted and displayed or used based on your needs
850000
First Word Binary Status i32
First Word 24 Element Bool Array
800000
Second Word Binary Status i32
Second Word 24 Element Bool Array
Bit 0
Check the PMAC Documentation for Bit meanings!
Bit 23

Depending on your needs the extraction of individual status bits is a tedious task. Later exercises introduce PMACPanel ICVs that extract and display the most common status bits for you thereby eliminating the tedium. The diagram for the example is shown here.



PmacTutor6 - Accessing PMAC I-Variables

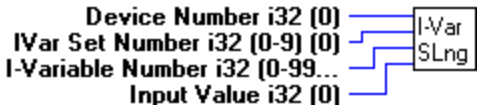
On PMAC, I-variables (Initialization, or Set-up, Variables) determine the personality of the controller for a given application. In general, this is a supervisory task. They are at fixed locations in memory and have pre-defined meanings. Most are integer values, and their range varies depending on the particular variable. There are 1024 I-Variables, from I0 to I1023, and they are organized as follows:

I-Variable Range	Functional Group
I0 -- I75	General card setup (global)
I76 -- I99	Reserved for future use

I100 -- I186	Motor #1 setup
I187 -- I199	Coordinate System 1 setup
I200 -- I286	Motor #2 setup
I287 -- I299	Coordinate System 2 setup
I3xx, I4xx, …	Motor #3, Coordinate System 3, …
I800 -- I886	Motor #8 setup
I887 -- I899	Coordinate System 8 setup
I900 -- I979	Encoder 1 - 16 setup (in groups of 5)
I980 -- I1023	Reserved for future use

To support this organization and facilitate access PMACPanel provides a special collection of VIs to manipulate and access them. Each type of I-Variable, Boolean, Short, Long, etc. has three VIs. The VIs for accessing Long (i32) I-Variables are used to illustrate the interface.

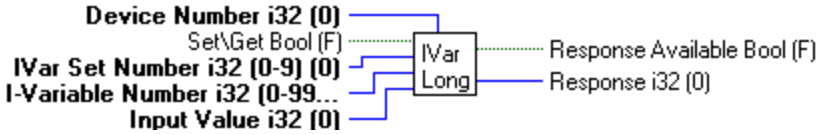
- **PmacIVarSetLong** - Set the Long I-Variable specified by IVar Set Number and I-Variable Number. The variable address is calculated as IVar Set Number * 100 + I-Variable Number. IVar Set Number = 0 addresses global I-Variables. IVar Set Numbers from 1 - 8 address motors and coordinate system I-Variables.



- **PmacIVarGetLong** - Get the Long I-Variable specified by IVar Set Number and I-Variable Number. The variable address is calculated as IVar Set Number * 100 + I-Variable Number. IVar Set Number = 0 addresses global I-Variables. IVar Set Numbers from 1 - 8 address motors and coordinate system I-Variables.

- **PmacIVarLong** - If Set\Get is FALSE or not wired get the Long I-Variable specified by IVar Set Number and I-Variable Number. Response Available will be TRUE to indicate Response contains the new value. If Set\Get is TRUE set the Long I-Variable using Input Value. Response Available will be FALSE and Response defaults to Input Value.

  The variable address is calculated as IVar Set Number * 100 + I-Variable Number. IVar Set Number = 0 addresses global I-Variables. IVar Set Numbers from 1 - 8 address motors and coordinate system I-Variables.



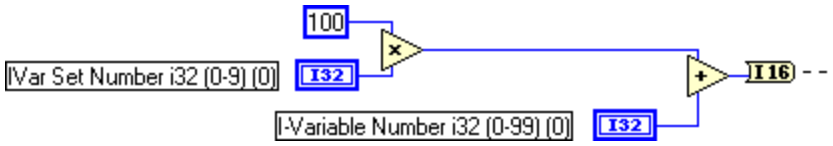
The first two I-Variable operations are obvious. The Get/Set VIs exists because when developing GUIs to configure I-Variables you want to get them for display and set them for modification. Grouping these operations together in a single VI simplifies your diagrams. Note that the Set\Get terminal is not required. If it is not wired the default operation for the VI is to Get the I-Variable. This type of Set/Get VI architecture is very common in PMACPanel.

Identical sets of VIs are provided for

- **PmacIVarDbl**, **PmacIVarGetDbl**, **PmacIVarSetDbl**
- **PmacIVarBool**, **PmacIVarGetBool**, **PmacIVarSetBool**
- **PmacIVarShort**, **PmacIVarGetShort**, **PmacIVarSetShort**

There are no string I-Variables. Many of the I-Variables are bit-mapped. ICVs for collecting I-Variables into functional groups and manipulating the bit-mapped I-Variables are introduced as required later.

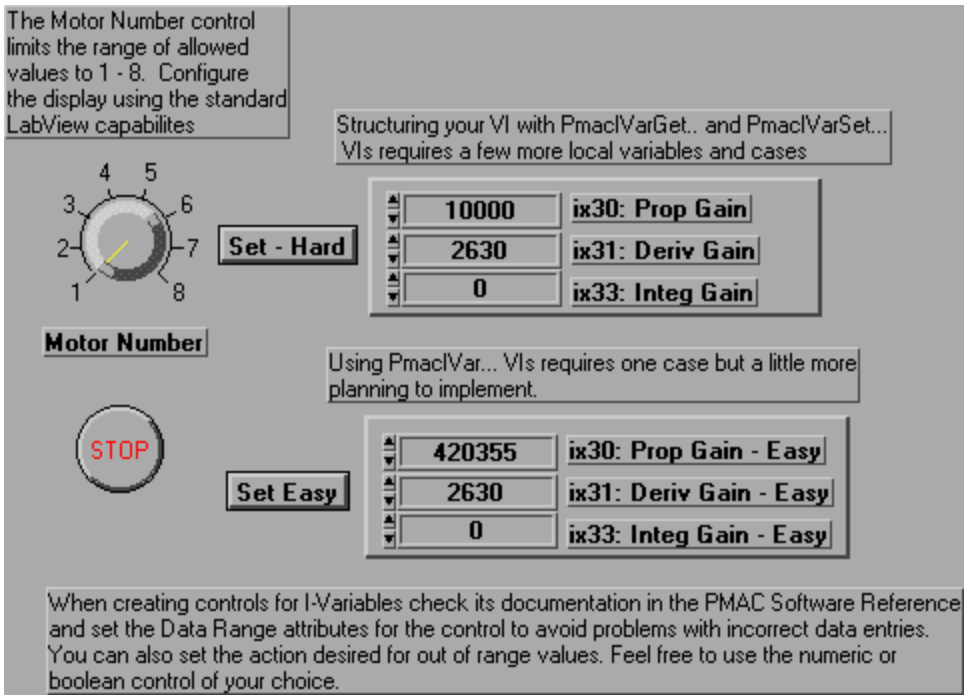
To access an I-Variable the I-Variable Set Number and I-Variable Number are used to compute the number of the requested I-Variable as shown.



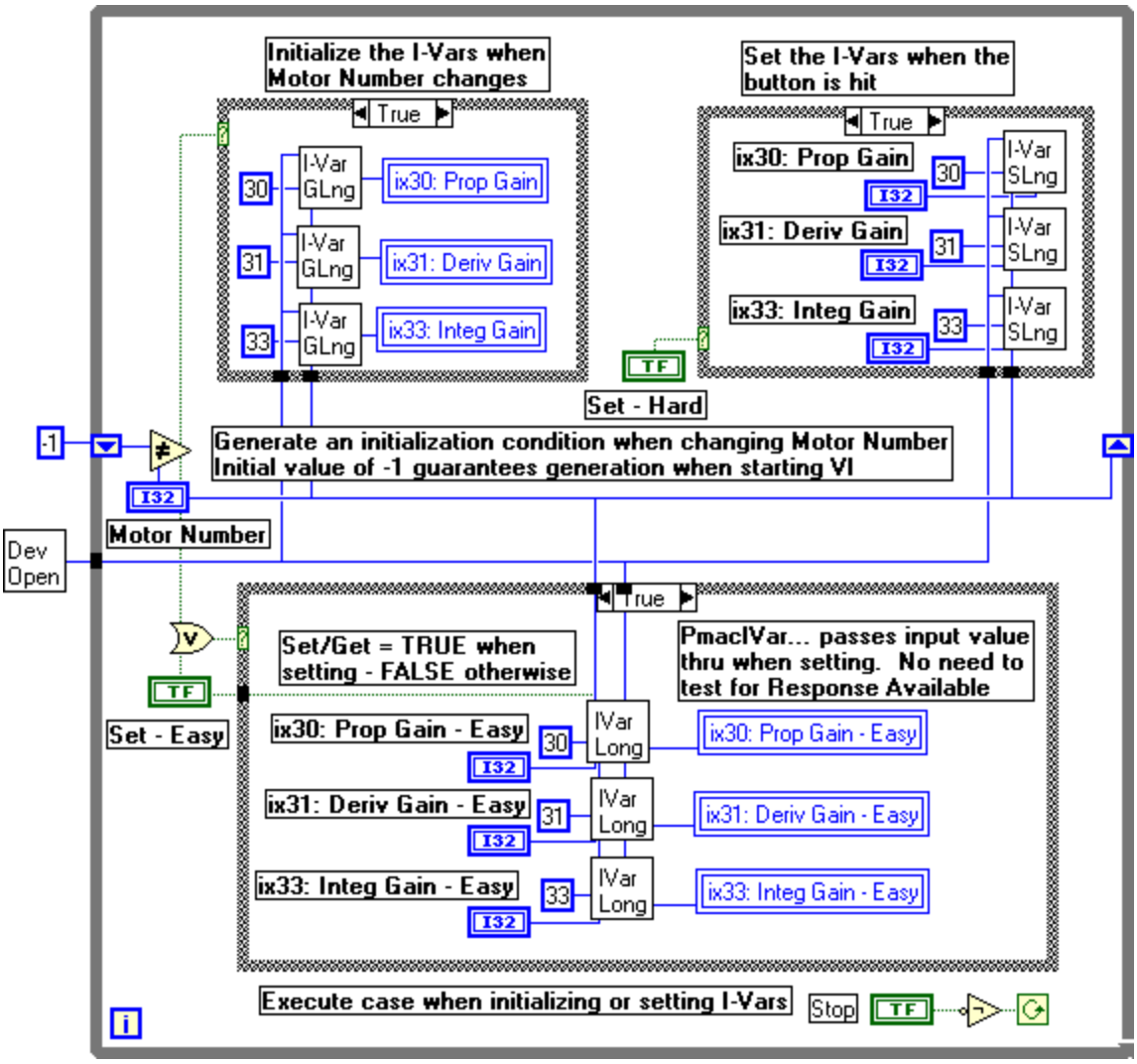
Using this approach, development of ICVs that manipulate collections of I-Variables for a particular motor or coordinate system is easy. PMACPanel does not check to see whether the I-Variable being addressed exists, its type, or its range. You can find this information in the *PMAC Software Reference Manual*. The I-Variable clusters introduced later perform this type of range checking where appropriate.

In addition to the organizational architecture I-Variables are accessed differently by PComm32. You can use the **PmacResponse** VIs introduced in **PmacTutor4** to access them. The problem is that depending on value of I9 I-Variable queries may be returned as decimal or hexadecimal values. PMACPanel VIs in the **PmacResponse** collection do not support the conversion of hexadecimal responses into numerical data types. If you use **PmacRespGetLong** to request an I-Variable like Ix25 you will get a response of zero if I9 = 2 or 3 - not what you want. If you use **PmacIVarGetLong** you get the proper binary representation and you get the ability to organize your access into groups.

The panel for this exercise allows you to modify a few Motor PID loop I-Variables. When creating your application's VI panels you should limit the range of controls to prevent potentially damaging data from being entered by the user.


The Motor Number control limits the range of allowed values to 1 - 8. Configure the display using the standard LabView capabilites
Structuring your VI with PmaclVarGet.. and PmaclVarSet... VIs requires a few more local variables and cases
4
5
3
6
2
7
1
8
Set - Hard
10000
ix30: Prop Gain
2630
ix31: Deriv Gain
0
ix33: Integ Gain
Motor Number
Using PmaclVar... VIs requires one case but a little more planning to implement.
STOP
Set Easy
420355
ix30: Prop Gain - Easy
2630
ix31: Deriv Gain - Easy
0
ix33: Integ Gain - Easy
When creating controls for I-Variables check its documentation in the PMAC Software Reference and set the Data Range attributes for the control to avoid problems with incorrect data entries. You can also set the action desired for out of range values. Feel free to use the numeric or boolean control of your choice.

The diagram for this VI demonstrates two ways of implementing I-Variable access. As a general principle, I-Variables should be read when your application begins and anytime you access a different group. In this example, when the user changes the Motor Number the application logic generates a Boolean condition indicating this and re-initializes the panel controls. When the user clicks an update button, the values contained in the controls are sent to PMAC.



The use of **PmacIVar…** as opposed to the **PmacIVarSet… and PmacIVarGet…** VIs groups the controls into sub-units that are a little more manageable. They are used extensively in the ICVs introduced later.

PmacTutor6b - Accessing PMAC Memory

PMAC makes extensive use of the Motorola 56K memory mapped architecture. This includes various encoder registers, DAC and ADC values, digital I/O ports, etc. Details of PMAC's memory organization can be found in the *PMAC Software Reference Manual* and should be consulted when accessing memory.

The **PmacMemory** collection of VIs simplifies your access and manipulation of this architecture and its binary representation. LabVIEW numerical controls and indicators can be configured to display this information as either hex or decimal data independent of the integer representation of the data. Data is actually received from PMAC and sent to PMAC in this collection using ASCII hexadecimal strings.

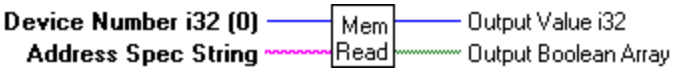
When defining a PMAC address to access the Address Spec String input to the following VIs can be in either hexadecimal or decimal form. Both strings below access the same address

```
Y:$C000
Y:49152
```

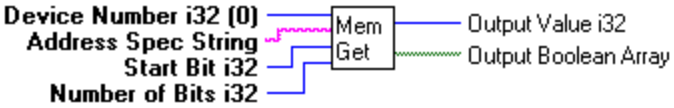
Reading Memory Data

There are two VIs to read and manipulate memory data in various forms. Remember that PMAC's integers are 24-bit words.

- **PmacMemoryRead** - Read a 24-bit quantity from the memory location specified by Address Spec String. For example, X:$002B. The result is output as both an i32 and a Boolean array.



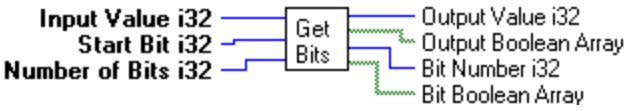
- **PmacMemoryGet** - Output Value is the value of the bit field defined by Start Bit and Number of Bits at the specified memory address. Output the field as both Output Value and Output Boolean Array.



The data retrieved from PMAC can be manipulated using the following VIs.

- **PmacMemoryGetBit** - Bit Value is the bit at Bit Number in Input Value..

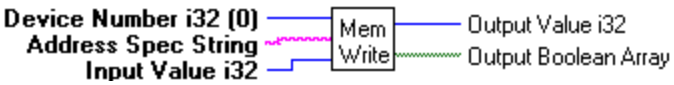
- **PmacMemoryGetBits** - Fetch the field defined by Start Bit and Number of Bits from Input Value. Return the field as Output Value i32, Output Boolean Array, and Bit Number (same as Output Value). Bit Boolean Array can be used with sets of radio buttons. If Output Value = 3 then Bit Boolean Array is 0, 0, 0, 1.



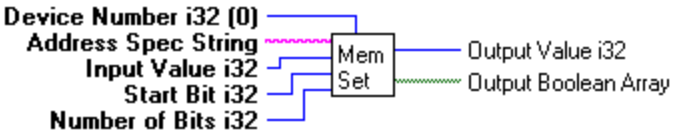
Writing Memory Data

There are two VIs to directly manipulate memory data in various forms. The data is first read from PMAC, modified, then rewritten.

- **PmacMemoryWrite** - Write a 24 bit quantity (Input Value) to the memory location specified by Address Spec String. For example, X:$002B. Pass Input Value to Output Value and Output Boolean Array.

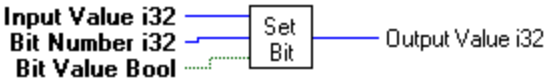


- **PmacMemorySet** - Write Input Value to a bit field defined by Start Bit and Number of Bits at the specified memory address. Output Value and Output Boolean Array are the value of the entire memory location with the new field.

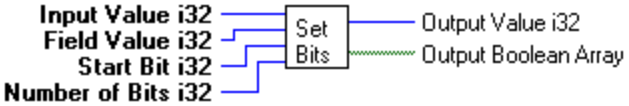


To implement these VIs the following two VIs are used to manipulate the data.

- **PmacMemorySetBit** - Set Bit Number in Input Value using Bit Value. The new word is Output Value.



- **PmacMemorySetBits** - Insert Field Value into Input Value at the field defined by Start Bit and Number of Bits. Output Value is Input Value with Field Value inserted. Output Boolean Array is the Boolean representation of Output Value.



Reading and Writing 48 Bit Memory Data

Double word (48-bit) memory data is handled differently than single word (24-bit) data. **PmacMemoryReadDbl** and **PmacMemoryWriteDbl** provide two representations of the data - native LabVIEW double and two i32 integers one for the Hi X word and one for the Lo Y word. You should not attempt to access bits using logical bitwise operations such as Value & 32 on the double representation. You can test them using logical comparison operations such as Value == 32. Bitwise operations on the Lo and Hi word are OK. Specifying addresses for double words must be done using the following notation

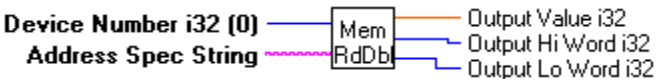
```
L:$002b
```

Specifying the address as

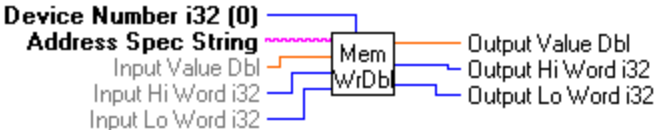
```
D:$002b
```

is not recognized by PMAC.

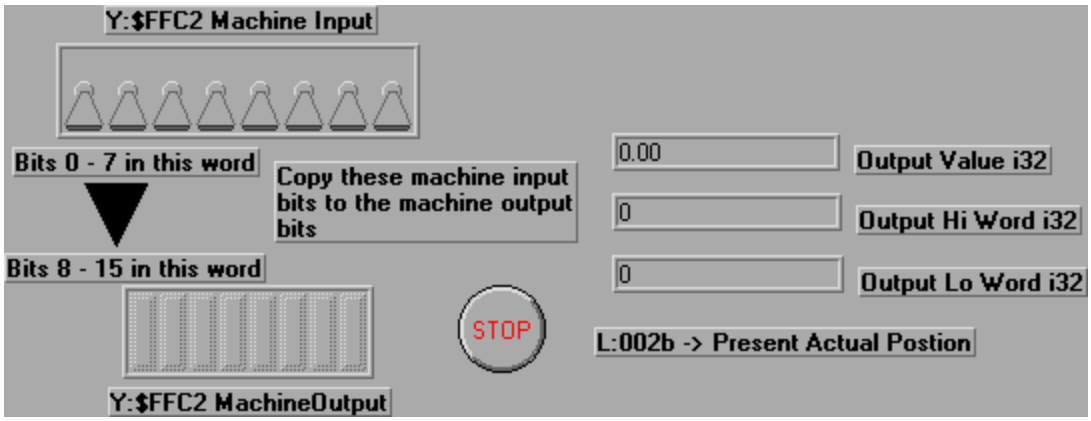
- **PmacMemoryReadDbl** - Read a 48-bit quantity from the memory location specified by Address Spec String. For example, L:$002B. The result is output as both a double and a hi-word and lo-word.



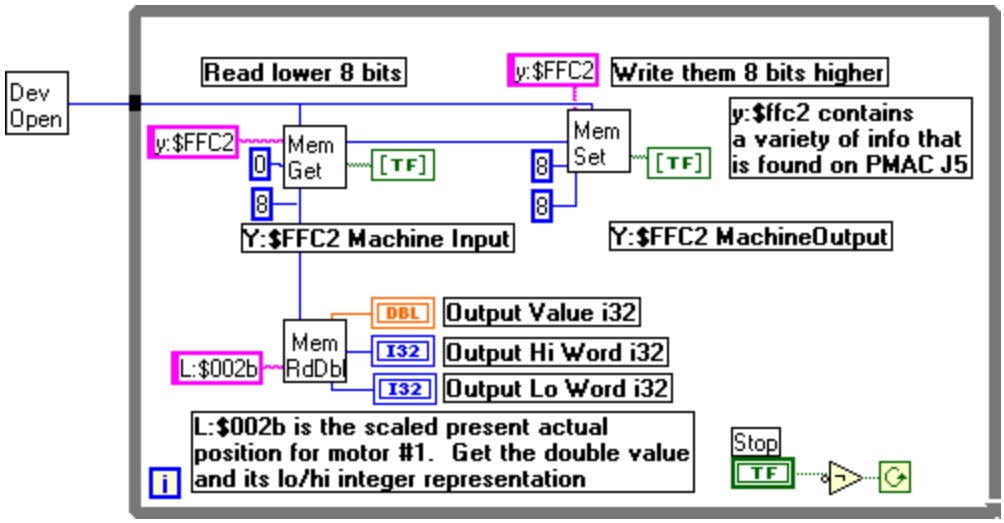
- **PmacMemoryWriteDbl** - Write a 48 bit quantity (Input Value) to the memory location specified by Address Spec String. For example, L:$002B. Input Value is copied to Output Value.



The panel for the exercise demonstrates the reading of a memory location containing the standard Machine Input at Y:$FFC2 to the standard Machine Output at Y:$FFC2. It also demonstrates accessing the 48-bit long word L:$002B that is the Present Actual Position for motor 1.



The diagram for this exercis e demonstrates how the lower 8 bits of Y:$FFC2 are written to the same memory location 8 bits higher.



PMACPanel ICVs

The previous set of exercises introduced you to PMACPanel's Device, Communication, and Query/Response interfaces to PMAC. They handle the details of sending commands and requests to PMAC and converting basic responses into LabVIEW data formats. The exercises in this section introduce another level of PMACPanel capabilities that provide indicators, controls, and VIs for many of PMAC's most common on-line commands. These VIs

- Handle queries for motor, coordinate system, and global commands
- Define common indicator and control clusters for use on your panels
- Implement function VIs for the indicators and controls

Using these as is, and modifying those that you desire, allows you to create great looking panels for your applications quickly. PMACPanel's ICV collections are organized into five categories.

- **PmacAcc**
- **PmacMotor**
- **PmacMotors**
- **PmacCoord**
- **PmacGlobal**

Each of these categories has several exercises to introduce its capabilities. You will also find similar examples in their respective sub-directories.

Each tutorial introduces an example and then selectively drills its way into supporting VIs. In doing so you, as the developer, will get a deeper understanding of PMACPanel internals so that you can address potential limitations in your design and enhance its capabilities to suit your specific requirements.

On-line Commands

PMAC provides a very large selection of on-line commands for monitoring and control. Any of these commands can be sent to PMAC using the VIs already introduced. Not every command is supported or used by the ICVs introduced here. Some are rarely used. Some should really be used from Pewn32. Others are potentially dangerous (O100 turns a motor on 100%!). Many would and some should never be used in a user application. A complete listing of available commands and their use can be found in the *PMAC Software Reference Manual*. The PMAC on-line commands used by PMACPanel VIs are listed here.

Global Commands

Addressing mode commands

& Report currently addressed CS

Buffer control commands

CLOSE	Close an open program buffer
DELETE GATHER	Return gather buffer space
LIST PLC	List a PLC in memory
LIST PROG	List a program in memory
SIZE	Return available buffer space

Control-character commands

<CONTROL A>	Abort all programs and moves
<CONTROL D>	Disable all PLC programs
<CONTROL F>	Report following errors for all motors
<CONTROL K>	Kill all motors
<CONTROL P>	Report position of all motors
<CONTROL Q>	Quit all executing motion programs
<CONTROL R>	Begin execution of motion programs in all coordinate systems
<CONTROL S>	Step working motion programs in all coordinate systems
<CONTROL V>	Report velocity of all motors

General global commands

$$$	Full card reset
$$$* **	Reinitialize PMAC to factory default
???	Report global status words
SAVE	Save current configuration to NOVRAM

Global variable commands

I, P, Q, and M	Variable access in numerous ways.

PLC control commands

DISABLE PLC	Disable a PLC program
ENABLE PLC	Enable a PLC program

Register access commands

R[H]{address}	Read data from memory
W{address}	Write data to memory

Coordinate System Commands

Axis definition commands

#{constant}->	Query PMAC for motor definition in CS

Buffer control commands

LIST PC, PE	List program at Program Execution

General CS commands

??	Report coordinate system status words

Program control commands

A	Abort program
B	Begin program
H	Hold program
Q	Quit program
R	Run program
S`	Step program

Motor Commands

General motor commands

$	Reset motor
HOME	Home
HOMEZ	Zero move home
K	Kill output

Jogging commands

J+	Jog positive
J^{constant}	Jog relative to actual position
J/	Jog stop
J-	Jog negative
J={constant}	Jog to position

J=	Return to pre-jog

Reporting commands

P	Report addressed motor position
V	Report addressed motor velocity
F	Report addressed motor following error
?	Report addressed motor status

PmacMotor ICVs

This series of exercises introduce the contents of the **PmacMotor** collection of ICVs. These allow your applications to add configuration, control, and monitoring for individual motors to your applications.

PmacTutor7 - Position, Velocity, Error, and Jogging

The most basic motion operations involve controlling or jogging motors under manual control and monitoring the position, velocity, and following error during the move.

Requesting and Formatting P, V, and E

PmacMotor has three VIs that request and format motor position, velocity, and following error for your use. These require a Coordinate Specify Cluster input. This cluster is more of a data type than a cluster associated with a specific control. It is often assembled from controls in your own application. It is defined as

Coord Specify Cluster Cluster defining the motor, CS, and conversion state to be applied

Coord Number i32 (1-8) (1) Coordinate number to use

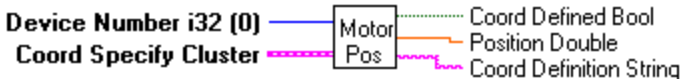
Motor Number i32 (1-8) (1) Motor Number to use

Convert Bool Apply a conversion for the specified motor in the specified CS

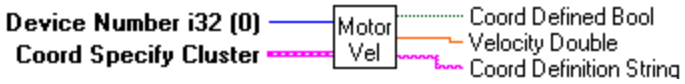
The VIs are:

- **PmacMotorPosition** - Query PMAC for Motor Number's position. PMAC reports the value of the actual position register plus the position bias register plus the compensation correction register, and if bit 16 of Ix05 is 1 (handwheel offset mode), minus the master position register.

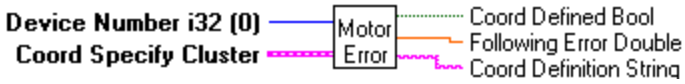
Coord Specify Cluster specifies a motor within a CS and an attempt to convert motor position from encoder counts to CS units. If the motor is not defined in the CS no conversion is applied. If the motor is defined and Convert is TRUE Coord Defined is TRUE and position is scaled from encoder counts to CS units. Coord Definition is a string specifying position units as "Encoder" or the CS definition of the motor.



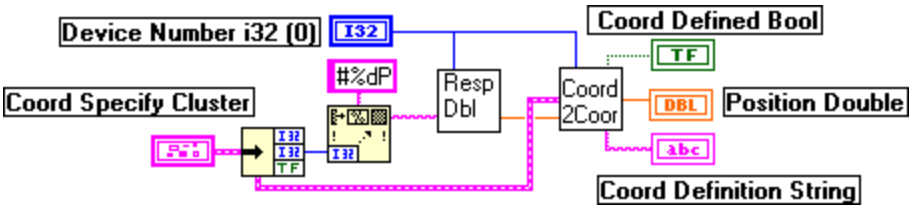
- **PmacMotorVelocity** - Query PMAC for Motor Number's present actual motor velocity, scaled in counts/servo cycle, rounded to the nearest tenth. The raw response reports the contents of the motor actual velocity register (divided by [Ix09*32]). This is converted to counts/msec by multiplying by 8,388,608 and dividing by the I10 default 3,713,707. If I10 is changed, modify this value in the diagram.



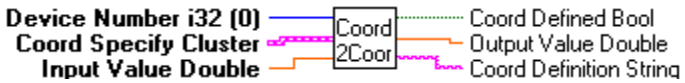
- **PmacMotorError** - Query PMAC for Motor Number's following error. Following error is the difference between motor desired and measured position at any instant. When the motor is open-loop (killed or enabled), following error does not exist and PMAC reports a value of zero.



The implementations of these VIs rely on the **PmacCoord** collection to convert values reported in mo tor encoder counts to coordinate system units. This capability is a fundamental component of PMACPanel. You will get tired of seeing the description of this process. The diagram for **PmacMotorPosition** shows that these VIs format a command string to request the desired motor position and let **PmacCoordMotor2Coord** process the response. The details of this implementation are left until later and gets involved.

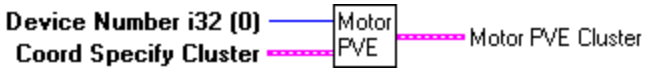


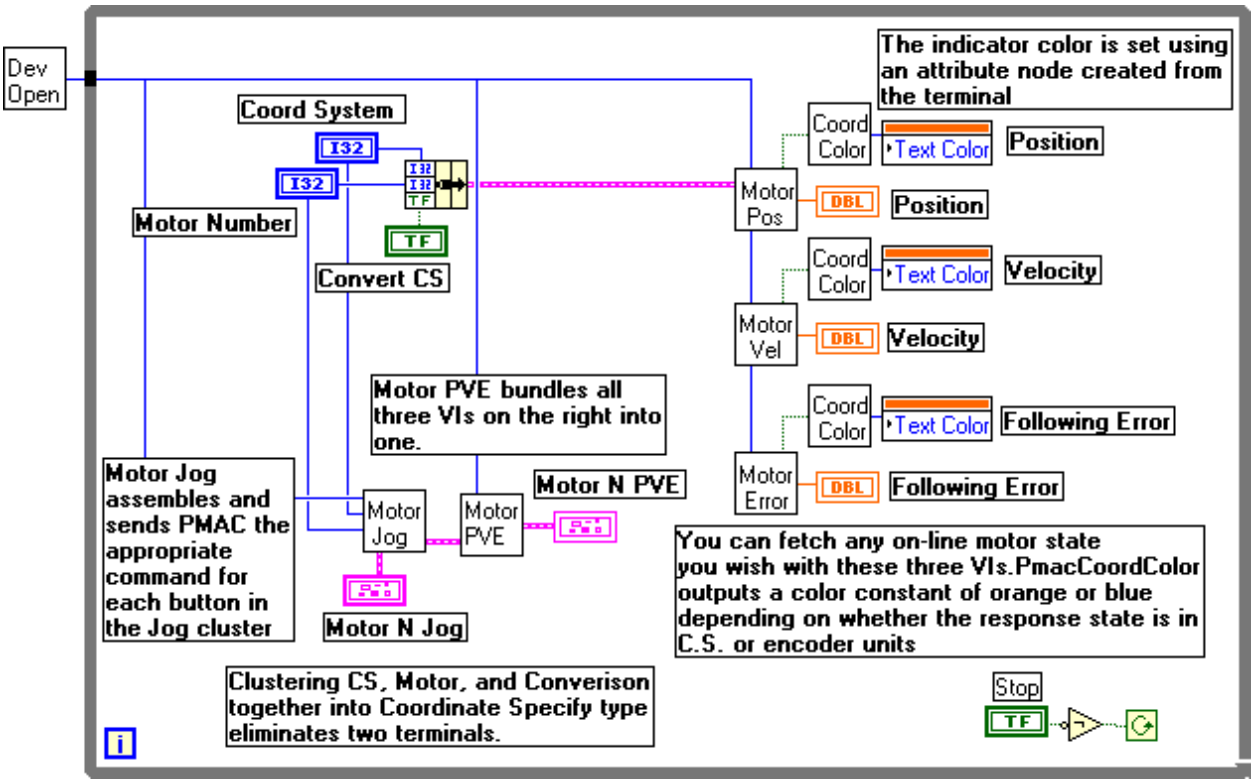
- **PmacCoordMotor2Coord** - Coord Specify Cluster specifies a motor within a CS and an attempt to convert Input Value from encoder counts to CS units. If the motor is not defined in the CS, no conversion is applied. If the motor is defined and Convert is TRUE Coord Defined is TRUE and Output Value is scaled from encoder counts to CS units. Coord Definition is a string specifying Output Value units as "Encoder" or the CS definition of the motor.



PmacMotorPVE is an extension of the position, velocity, and following error VIs that combine them into a single VI that can be used to drive the **PmacMotorPVE** indicator. The use of these VIs is shown in the diagram below.

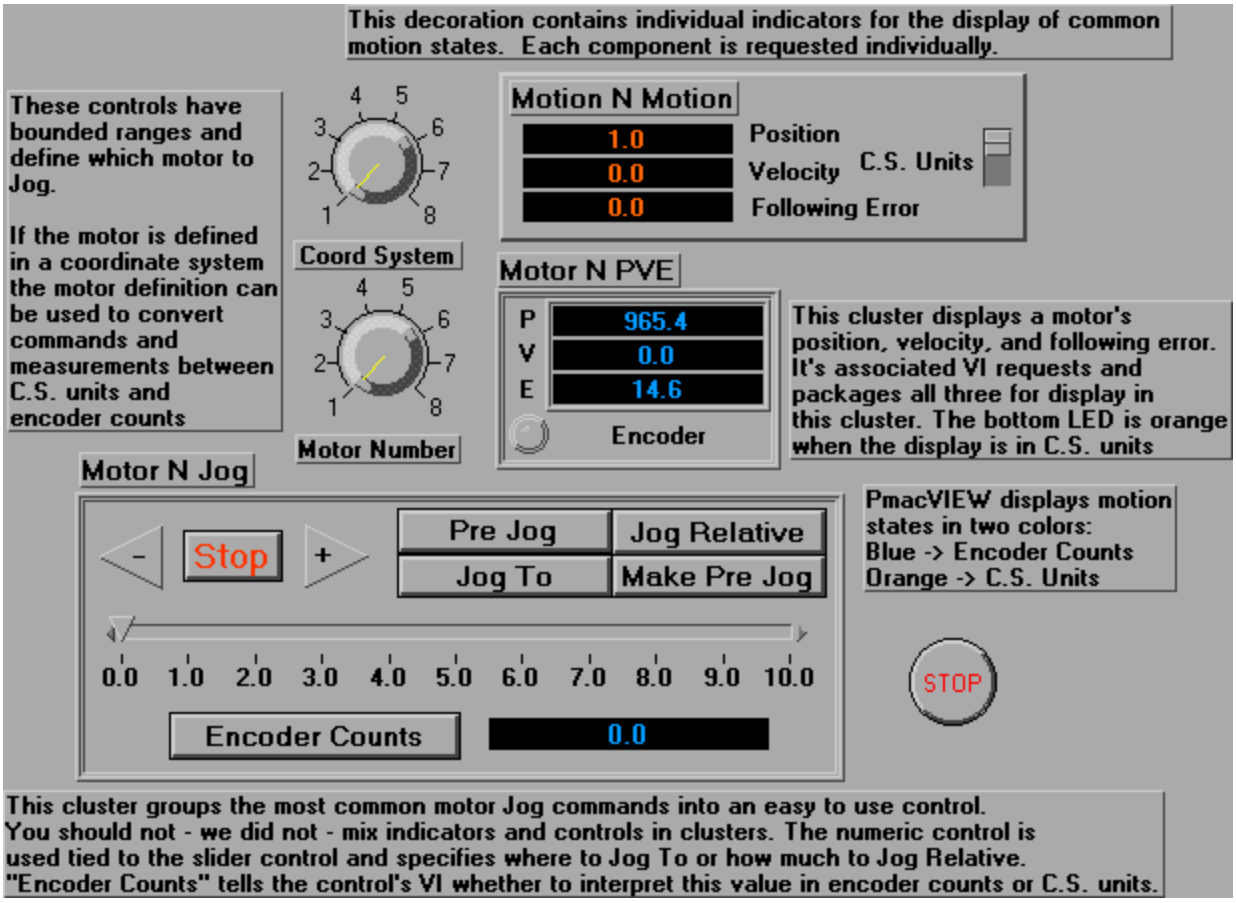
- **PmacMotorPVE** - Query PMAC for the position, velocity, and following error for Motor Number operating in Coord Number. Assemble the measurements into Motor PVE Cluster. If Convert is TRUE convert the measurements to CS units. Otherwise, leave them in encoder counts. See the documentation for PmacMotorPosition, PmacMotorVelocity, and PmacMotorError for details on how these individual values are produced.





A Coordinate Specify Cluster is assembled from three controls on the panel. Each of the individual Position, Velocity, and Error VIs receives the cluster. The P, V, and E VI outputs are used to drive three individual numeric indicators. In this example, the indicators have an attribute node to set their color. The color for the text is provided by **PmacCoordColor**. The indicator color is Orange if the reported value is in CS units and Blue if it is in encoder counts.

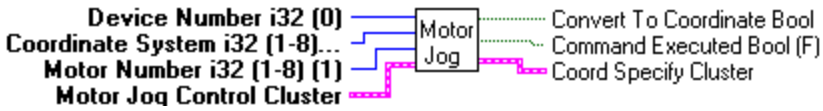
The panel for this exercise shows the indicators and the Boolean used to specify the units for P, V, and E displays.


This decoration contains individual indicators for the display of common motion states. Each component is requested individually.
These controls have bounded ranges and define which motor to Jog.
If the motor is defined in a coordinate system the motor definition can be used to convert commands and measurements between C.S. units and encoder counts
Coord System
Motor Number
Motor N Motion
1.0
0.0
0.0
Position
Velocity
Following Error
C.S. Units
Motor N PVE
P
V
E
965.4
0.0
14.6
Encoder
This cluster displays a motor's position, velocity, and following error. It's associated VI requests and packages all three for display in this cluster. The bottom LED is orange when the display is in C.S. units
Motor N Jog
Stop
Pre Jog
Jog Relative
Jog To
Make Pre Jog
0.0 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0
Encoder Counts
0.0
STOP
PmacVIEW displays motion states in two colors:
Blue -> Encoder Counts
Orange -> C.S. Units
This cluster groups the most common motor Jog commands into an easy to use control.
You should not - we did not - mix indicators and controls in clusters. The numeric control is used tied to the slider control and specifies where to Jog To or how much to Jog Relative.
"Encoder Counts" tells the control's VI whether to interpret this value in encoder counts or C.S. units.

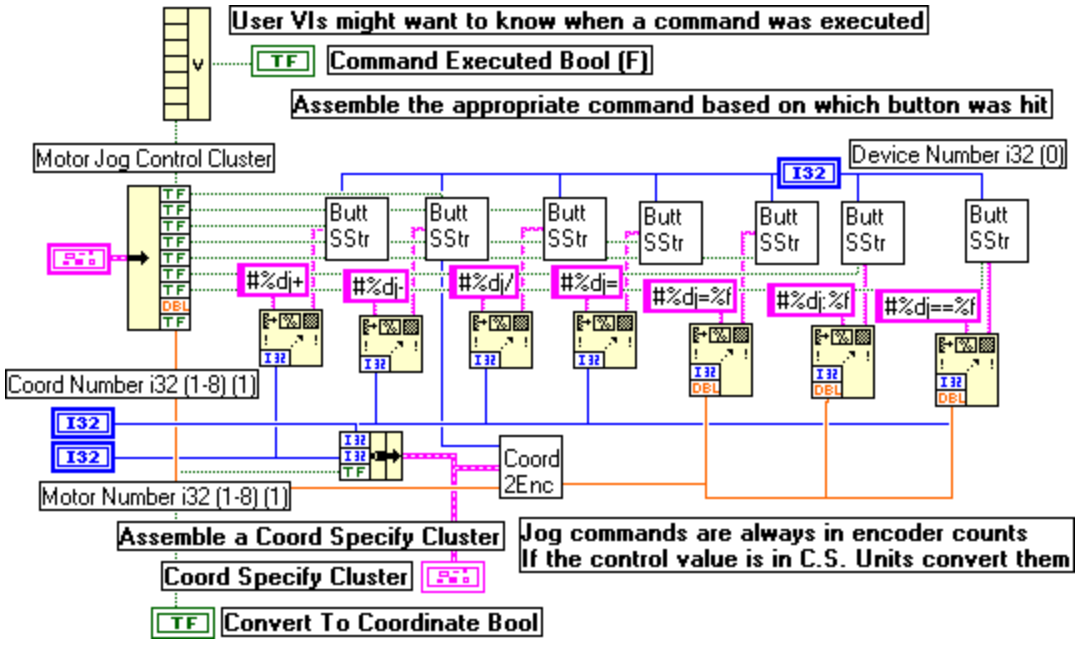
Generating On-Line Jog Commands

The front panel contains a cluster of controls defined by **PmacMotorJogControl** to group the most commonly used jog operations into a single panel item. This cluster must be used with its associated function VI to generate jog commands for PMAC.

- **PmacMotorJogControl** - Generate PMAC on-line commands for controlling jogging Motor Number. Command Executed Bool is TRUE when any button is clicked in Motor Jog Cluster. The value in the numeric control specifies the position Jog To and Jog Relative jog the motor to. This value is interpreted as either Encoder Counts (Default) or Coordinate Units in Coord Number as specified on the button. The button state is provided as the output Convert To Coordinate. This VI builds a Coord Specify Cluster using the various inputs to simplify the interface to PmacMotorPVE and other PMACPanel ICVs.



When buttons in the cluster are clicked the appropriate on-line command is assembled and sent to PMAC. The diagram for this VI illustrates the general architecture PMACPanel uses to generate on-line commands from control clusters.


User VIs might want to know when a command was executed
Command Executed Bool (F)
Assemble the appropriate command based on which button was hit
Motor Jog Control Cluster
Device Number i32 (0)
Butt SStr
#%dj+
#%dj-
#%dj/
#%dj=
#%dj=%f
#%dj:%f
#%dj==%f
Coord Number i32 (1-8) (1)
Motor Number i32 (1-8) (1)
Coord 2Enc
Assemble a Coord Specify Cluster
Jog commands are always in encoder counts
If the control value is in C.S. Units convert them
Coord Specify Cluster
Convert To Coordinate Bool

If you generate Motor or C.S. specific commands always send the Motor or C.S. number along with the command.

There are a few things to note about the organization of this VI. Unbundled command buttons are used in conjunction with a string format VI and **PmacButtSendStr** to create and send an appropriate command to PMAC. The position parameter required by some commands is converted into encoder units if the actual control value is in C.S. units. Using this architecture eliminates many case structures and allows you to add or delete commands as required.

Using **PmacMotorJogControl** and its cluster with **PmacMotorPVE** and its cluster you can quickly add motor jogging and position monitoring on any panel.

Control Clusters and Local Variables

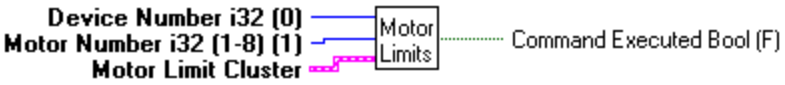
It has already been noted that clusters should contain either indicators or controls but not both. This is a generally bad idea in LabVIEW and because PMACPanel uses latched Boolean controls in most of its control clusters an extremely bad idea. When a VI reads the state of a latched Boolean control the control is reset. Hence, if there were two users of the Boolean state the second one to read the state would get the wrong answer. A corollary to this rule is that you cannot use local variable copies of control clusters that contain latched Boolean controls.

PmacTutor8 - Motor Control with Status Monitoring

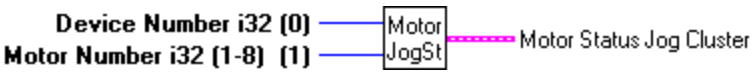
This exercise introduces another **PmacMotor** control and several indicator clusters for displaying motor status. As with the **PmacMotorJogControl**, status clusters require an associated VI to query PMAC for the required information and assemble it into a useful form.

This is the first exercise to actually generate an application you can really use to exercise PMAC. The panel contains **PmacMotorJogControl** and **PmacMotorPVE** already introduced. In addition, there is a **PmacMotorLimitControl** and two new status clusters. The VIs for the new panel are

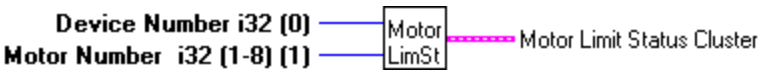
- **PmacMotorLimitControl** - Generate PMAC on-line commands for controlling move limits and operation for Motor Number. Command Executed Bool is TRUE when any button is clicked in Motor Limit Cluster.

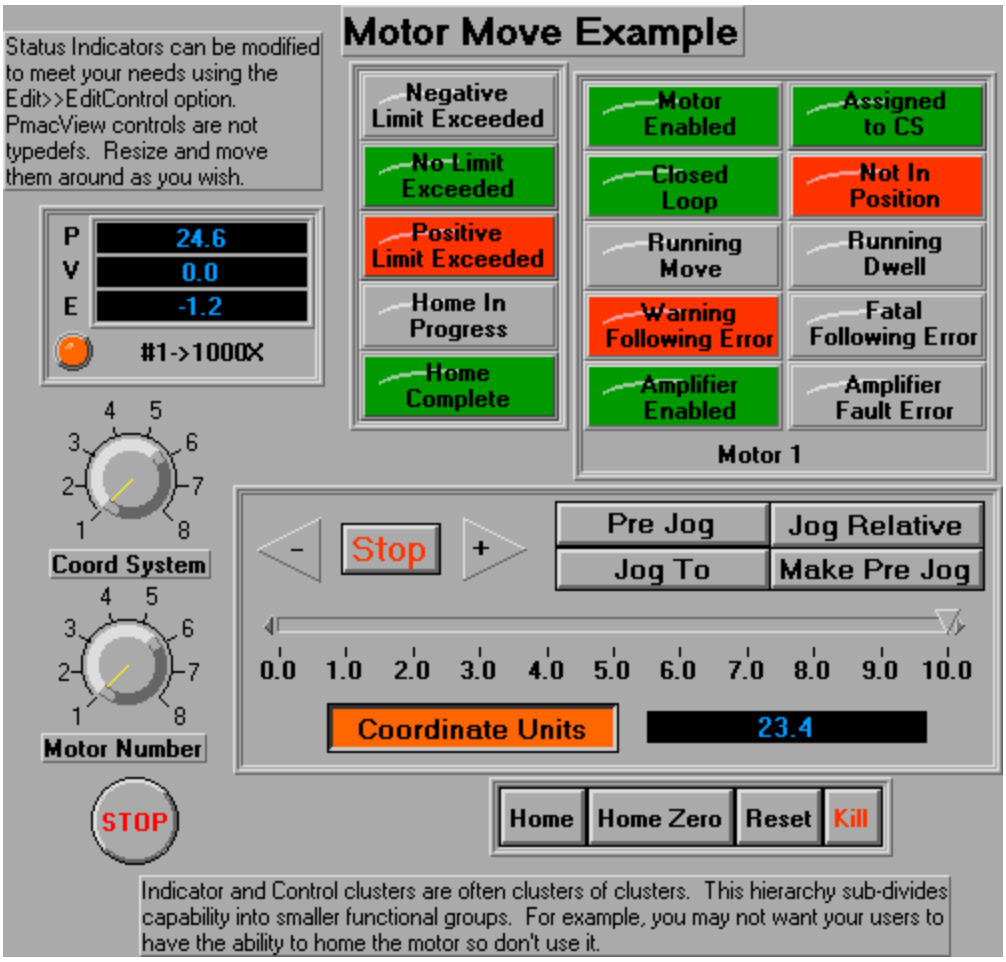


- **PmacMotorStatJog** - Create a status indicator cluster for the PmacMotorStatJog indicator containing the status for Motor Number.



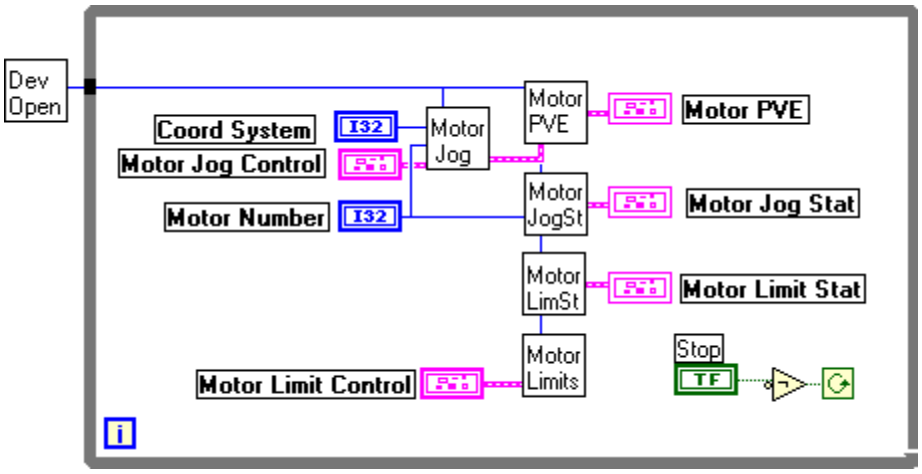
- **PmacMotorStatLimit** - Create a status indicator cluster for the PmacMotorStatLimit indicator containing the status for Motor Number.





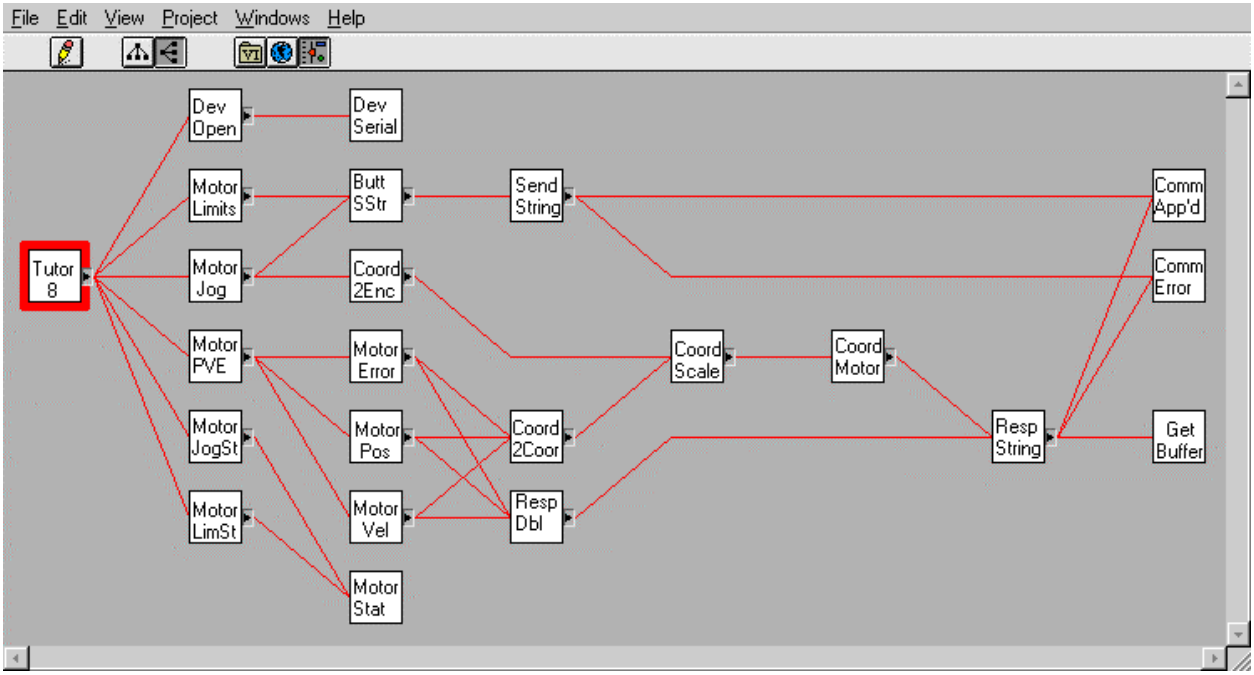
This panel, captured while actually running, demonstrates how useful these new capabilities are. The jog control indicates a move to a position of 23.4 defined in Coordinate Units not motor encoder counts. The PVE indicator has an Orange LED indicating that the displayed position, velocity, and following errors are in CS units. Furthermore, because the Coord System dial is set to 1, that motor 1 is defined as #1->1000X in CS 1. The status indicators show that Motor 1 is Assigned to a CS, Enabled, and in Closed Loop mode. Furthermore, that it is Not In Position and there is a Warning Following Error. You will note that the position in the Jog Control is specified as 23.4 CS Units and the PVE indicator shows an actual position of 24.6 CS Units. If you are actually running this exercise turn on Help by selecting **Help»Show Help**. As your cursor moves over the various indicator and control items in the clusters detailed help for each panel item is displayed.

The diagram for this panel is very simple because PMACPanel handles all the details for you. There are two control clusters, one PVE cluster, and two status clusters each with its associated function VI. Add two dials and you've created Jog application. You will note that the Coord Specify Cluster requires by **PmacMotorPVE** is constructed by **PmacMotorJog**.



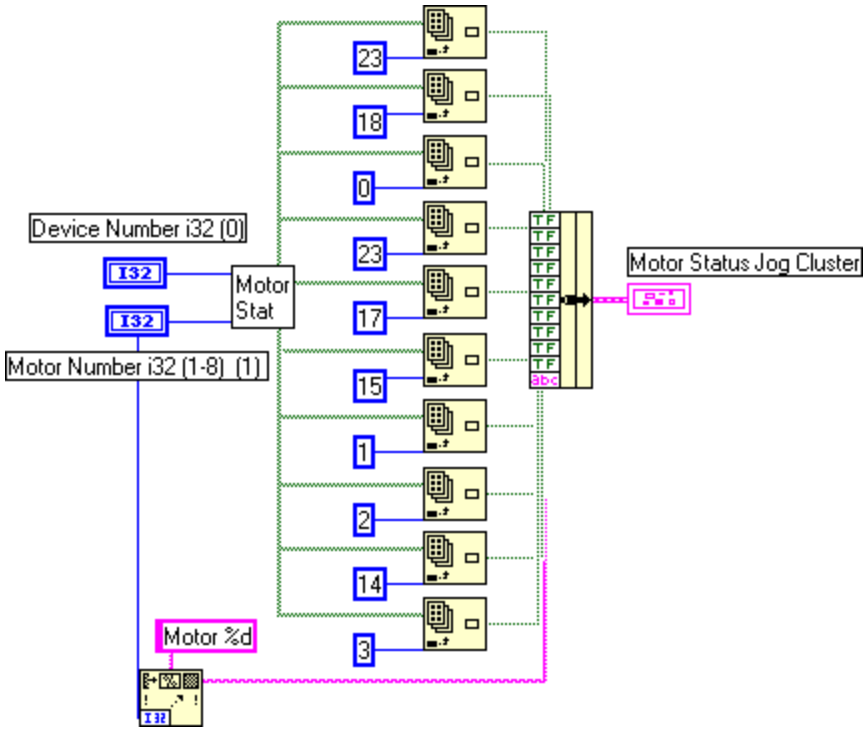
Hierarchical Encapsulation

PMACPanel attempts to break panel clusters and function VIs into manageable chunks that group functionality. Using this approach, you can piece together those items you need to build your application. Each VI builds on top of the capabilities provided by still lower levels until almost everything funnels through **PmacCommSendStr** and **PmacCommRespStr**. In using proper program design, the result is easier to maintain and modify for your own purposes. The VI Hierarchy for this exercise is shown here to illustrate this point.



Accessing Status Bits

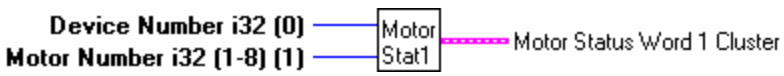
To illustrate how PMACPanel handles status information the diagram for **PmacMotorStatJog** is shown here. The VI calls the **PmacMotorStat** VI covered in **PmacTutor5** and the Boolean array for each status word is indexed to get the desired bit. The individual bits are assembled into a cluster for use by the indicator. Notice that a string is created indicating the associated motor in the status cluster. It automatically updates the indicator cluster so that you don't have to. If you don't want this simply eliminate it from the cluster and modify the panel cluster.

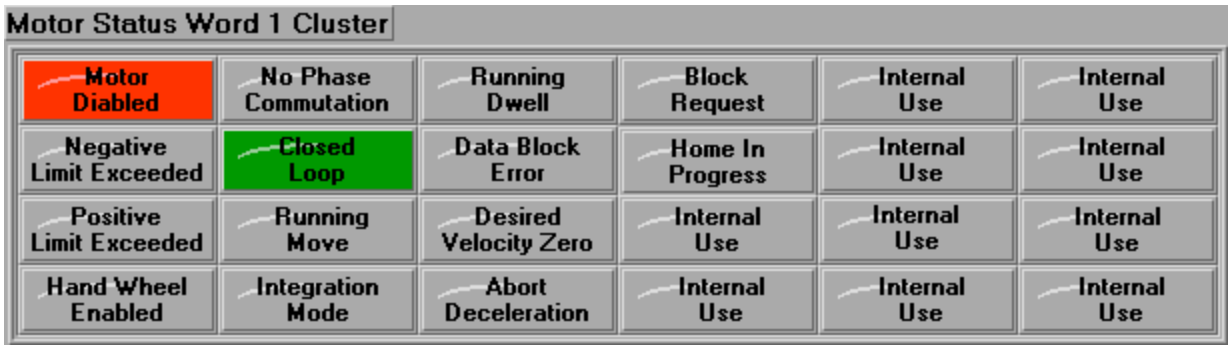


Motor Status VIs

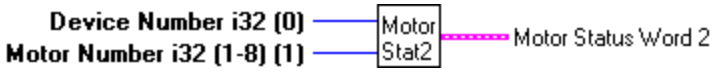
For completeness, PMACPanel provides indicator/VI pairs to monitor all bits in both motor status words. You can copy the individual cluster items into your own clusters to avoid creating them from scratch. Each cluster LED has been carefully constructed to include detailed help, labeling, and Boolean text. The icons along with the panel clusters are

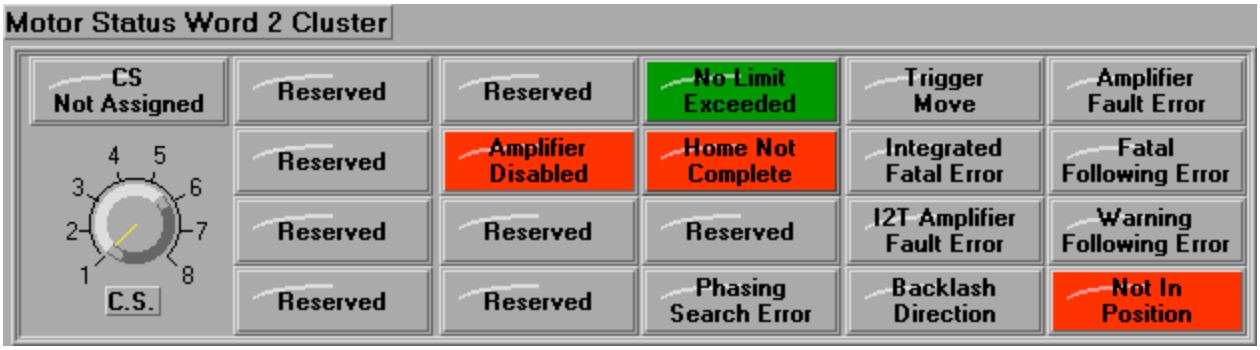
- **PmacMotorStat1** - Create a status indicator cluster for the PmacMotorStat1 indicator containing the status for Motor Number.





- **PmacMotorStat2** - Create a status indicator cluster for the PmacMotorStat2 indicator containing the status for Motor Number.





These clusters are organized to reflect the 4-bit organization of the status words. The first column is for the first four bits (23-20), the second column for the second four (19-16), etc. You should note that some bits are not defined hence they are Reserved or For Internal Use. Also, note that for motor status word 2 three bits are interpreted as the coordinate system to which the motor is assigned.

A Word on Status Indicator Colors

Human factor considerations play a major role in how you assign colors to your application's status indicators. Is Green good? Is it TRUE? If Green is TRUE and Red is FALSE then an Amplifier Fault is Green although it is probably not good. Setting up your definitions can be very confusing.

In PMACPanel status indicators attempt to convey a generally useful meaning by the LED's color and text. To clarify this a few examples are covered in more detail. You should feel free to change colors and text to reflect the intent within your application.

- **Following Errors:** These are Red when there is an error. For these status bits that means the bit is TRUE. When the bit is FALSE the indicators are Gray - there is no error. Gray can generally be interpreted as NOT TRUE

or not dangerous. The text in the indicator says what the indicator is - "Warning Following Error" either way.

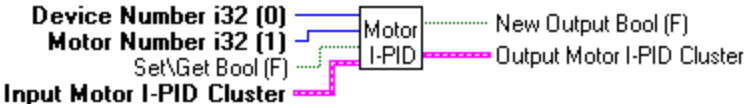
- **In Position:** This indicator is Red when the motor is Not In Position - the bit is FALSE. Its text says - "Not In Position". The indicator is Green and says "In Position" when the motor is in position - the bit is TRUE. Imagine a situation where motor Not In Position means its moving, therefore your program is running, and that is good. Do you now make the indicator Green?

- **Home in Progress:** This is Gray when there is no Home in Progress. This doesn't mean that the motor has been homed. When the motor is homing, the indicator is Green. It may well be that you want this to be Red when the motor is homing. The choice is yours.

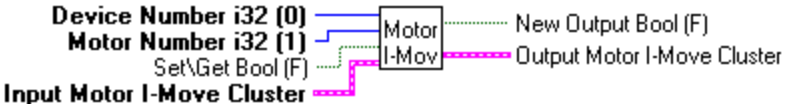
PmacTutor9 - Motor I-Variable Configuration

In **PmacTutor6,** the general architecture for developing supervisory VIs for modifying individual I-Variables was introduced. In this exercise four VIs to encapsulate the most common motor I-Variables are introduced. This type of hierarchical I-Variable architecture is used throughout PMACPanel.

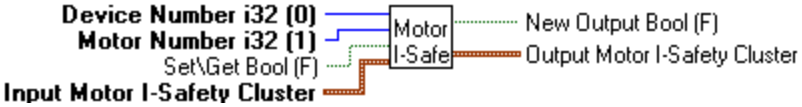
- **PmacMotorIVarPID** - Follow PMACPanel's standard I-Variable VI architecture. When Set/Get is TRUE the PID I-Variables for the specified Motor Number are set. Otherwise they are fetched from PMAC and provided by Output Motor I-PID Cluster with New Output TRUE. Set/Get is not required and defaults to a Get operation.



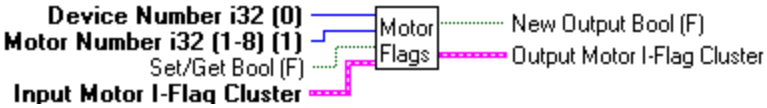
- **PmacMotorIVarMove** - Follow PMACPanel's standard I-Variable VI architecture. When Set/Get is TRUE the movement I-Variables for the specified Motor Number are set. Otherwise they are fetched from PMAC and provided by Output Motor I-Move Cluster with New Output TRUE. Set/Get is not required and defaults to a Get operation.



- **PmacMotorIVarSafety** - Follow PMACPanel's standard I-Variable VI architecture. When Set/Get is TRUE the safety I-Variables for the specified Motor Number are set. Otherwise they are fetched from PMAC and provided by Output Motor I-Safety Cluster with New Output TRUE. Set/Get is not required and defaults to a Get operation.

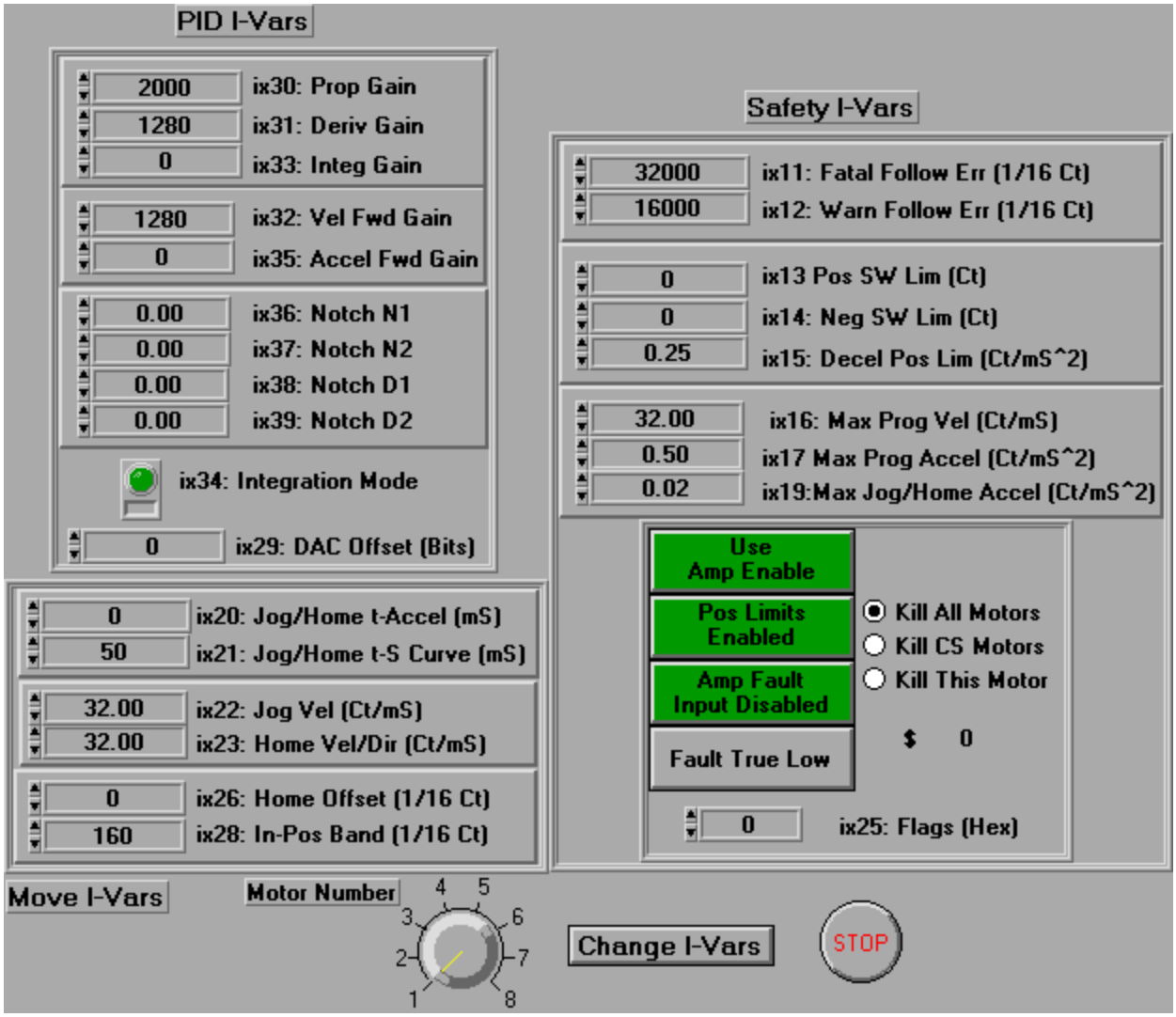


- **PmacMotorIVarFlag** - Follow PMACPanel's standard I-Variable VI architecture. When Set/Get is TRUE the encoder flag I-Variable Ix25 for the specified Coord Number are set. Otherwise they are fetched from PMAC and provided by Output Coord I-Flag Cluster with New Output TRUE. Set/Get is not required and defaults to a Get operation.

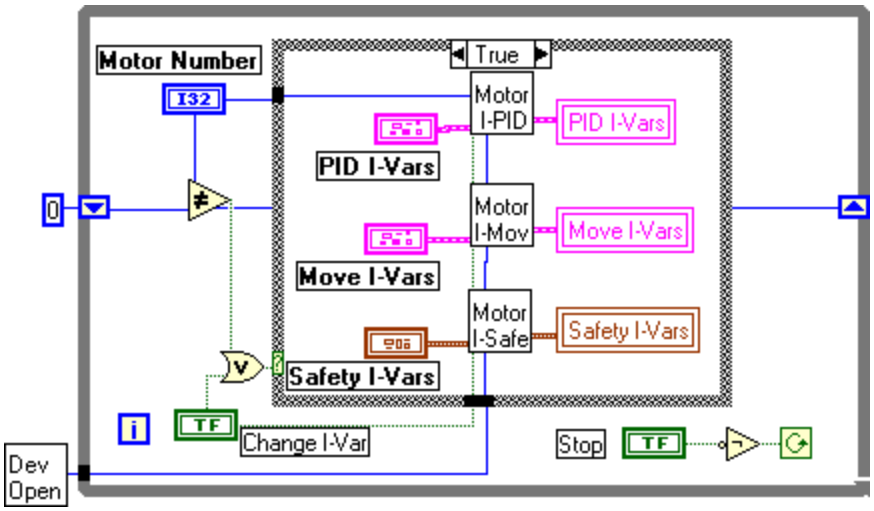


The panel for this exercise shows the three main **PmacMotorIVar** clusters. **PmacMotorIVarFlag** is actually a sub-cluster in **PmacMotorIVarSafety**.

To change the I-Variables click the Change I-Vars button. Each item in the cluster has a full description that is accessible using **Help»Show Help**. Because we have defined each item in the clusters as referencing a specific I-Variable, we have specified the item units and appropriate data ranges for each item



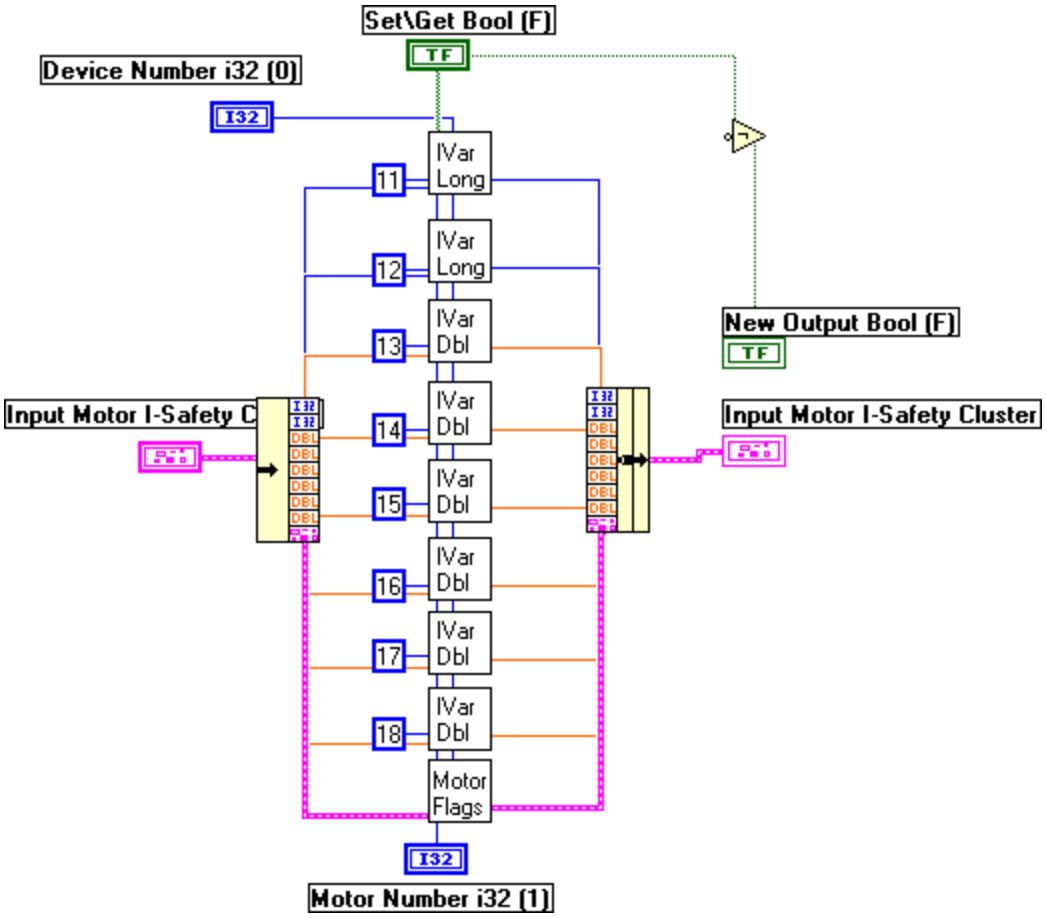
The diagram for the exercise demonstrates why using these ICVs makes life easier. The Boolean criteria for executing the case are the same as in **PmacTutor6**. It is executed whenever Motor Number is changed or the button is clicked. When the case is executed because of a change in Motor Number, the VIs perform a get operation and refresh the cluster contents with the configuration of the new motor.



The observant reader will note that the VI makes use of local copies of the panel control/indicator clusters. Those clusters that contain Boolean controls like **PmacMotorIVarPID** and **PmacMotorIVarFlag** have their Booleans set for Pressed mechanical action not latched action. They are not used to initiate a command or action. This doesn't violate the basic caution on using local copies of control clusters noted in this document in several places.

Grouping Multiple I-Variables

It should now be obvious why the **PmacIVar**… VIs require an I-Var Set Number. It allows them to be grouped by motor and/or coordinate system. To accomplish this it is only necessary to extend the concept to a slightly higher level. The diagram for **PmacMotorIVarSafety** is shown here. You should note that when one I-Variable is set they are all set whether or not they have changed. Therefore, the contents of the cluster should be refreshed by a Get operation prior to changing individual items and performing a Set operation. Controls and indicators for your panels should have the appropriate type and range defined to prevent inadvertent user inputs.



PmacMotors ICVs

This series of exercises introduce the **PmacMotors** collection of ICVs. These allow your applications to monitor and plot the motion of multiple motors.

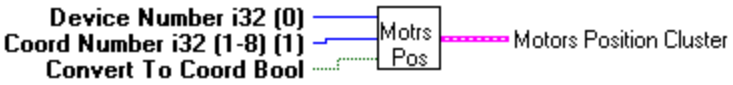
PmacTutor10 - Requesting and Plotting Motor Motion

This exercise introduces a number of indicators, controls, and VIs for requesting the motion of all motors. The data can be displayed on a PMACPanel cluster indicator, plotted in a strip chart, or analyzed using LabVIEW's extensive analysis capabilities. VIs for setting plot legends and selecting which motors to plot can be used to create flexible interfaces.

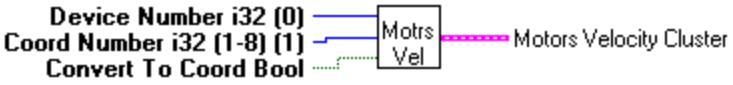
The primary query VIs in the collection request PMAC position, velocity, and following errors for all motors. They are not based on their counterparts in **PmacMotor**. These are

- **PmacMotorsPositions** - Query PMAC for the positions for all motors. PMAC reports the value of the actual position register plus the position bias register plus the compensation correction register, and if bit 16 of Ix05 is 1 (handwheel offset mode), minus the master position register.

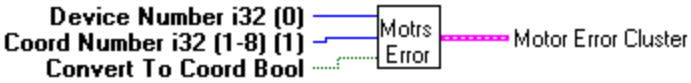
Assemble the measurements into **PmacMotorsPVE** Cluster. If Convert To Coord is TRUE convert the measurements to CS units for those motors defined in the CS. Otherwise, leave them in encoder counts.



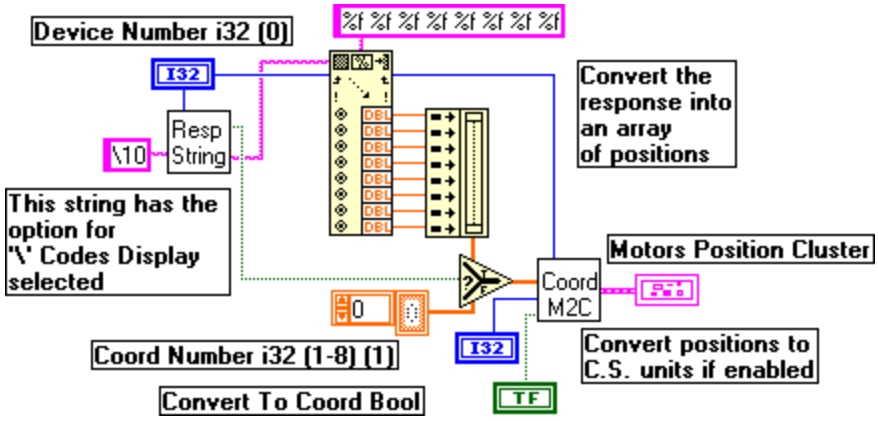
- **PmacMotorsVelocities** - Query PMAC for all motor's present actual motor velocity, scaled in counts/servo cycle, rounded to the nearest tenth. The raw response reports the contents of the motor actual velocity register (divided by [Ix09*32]). This is converted to counts/msec by multiplying by 8,388,608 and dividing by the I10 default 3,713,707. If I10 is changed, modify this value in the diagram.



- **PmacMotorsErrors** - Query PMAC for the following errors for all motors. Following error is the difference between a motor's desired and measured position at any instant. When a motor is open-loop (killed or enabled), following error does not exist and PMAC reports a value of 0.



The diagram for **PmacMotorsPositions** shows that a command string is sent requesting the position. This command is a control code and requires the proper option be set for the string constant. The response will have as many positions as there are motors in PMAC. The responses are converted into an array and processed by **PmacCoordMotors2Coord**. Which assembles the results into a cluster for display on a **PmacMotorsPVE** cluster. This is different from a **PmacMotorPVE** cluster



The **PmacMotorsPVE** cluster is comprised of an array of values, a Boolean array to indicate which motors have been converted to C.S. units, and a text

label for the cluster indicating which C.S. the display is using for those motors displayed in C.S. units.

Motors PVE Cluster The indicator cluster displays an array of values for all PMAC motors. The array may be positions, velocities, or following errors. The array of Boolean indicators indicate which values are in CS units. The caption specifies the displayed values as being in encoder counts or a specific CS.
See the documentation for PmacMotor (s)Position, PmacMotor(s)Velocity, and PmacMotor(s)Error for details on how these individual values are produced.

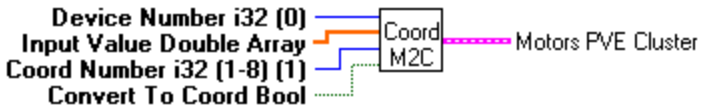
Motor Value Array Array of numerics for positions, velocities, or following errors for each motor. See the documentation for PmacMotor(s) Position, PmacMotor(s)Velocity, and PmacMotor(s)Error for details on how these individual values are produced.

C.S. Defined Array of Booleans indicating which motors are displayed in CS Units.

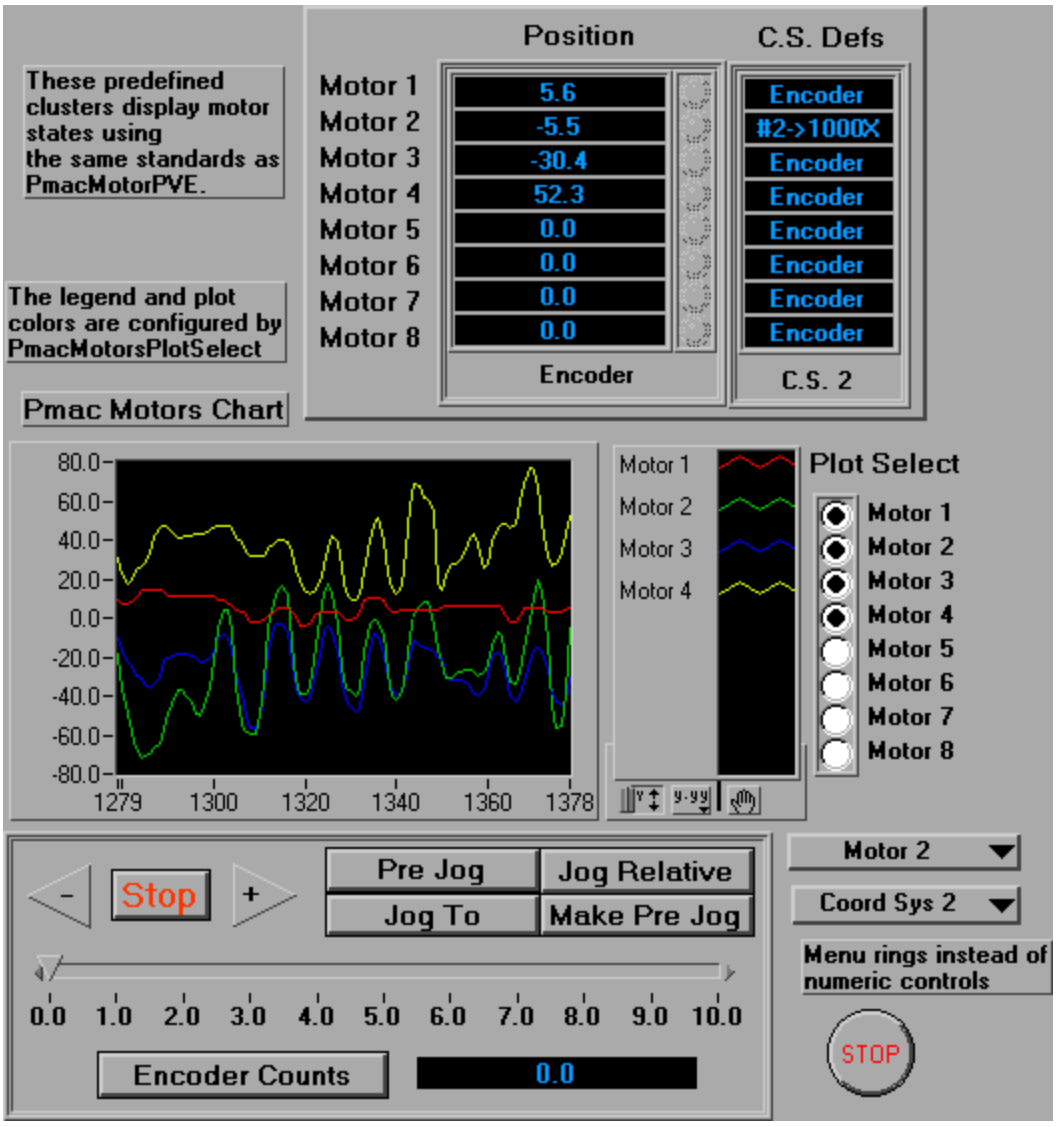
C.S. Applied Caption indicating the currently addressed coordinate system or that the displayed values are in Encode Counts.

The conversion of motor states from encoder counts to CS Units operates similar to **PmacCoordMotorToCoord** introduced in **PmacTutor7**. It is

- **PmacCoordMotorsToCoord** - Generate an indicator cluster for PmacMotorsPVE. Input Value Double is an array of positions, velocities, or following errors from VIs in the PmacMotors collection. If Convert To Coord is TRUE fetch the CS definitions for the motors specified in Coord Number and scale them to CS units. Motors not defined in Coord Number are not scaled.



The panel for the exercise uses the familiar jog control with LabVIEW menu rings for selecting the motor number and coordinate system number. The **PmacMotorsPVE** cluster and a C.S. definition cluster display the motor positions and motor definitions in the addressed CS. The plot is a standard strip chart with Auto Scaling on the Y-Axis. To plot one or more motors click the appropriate radio button. The colors for the plots and the legend are automatically updated.

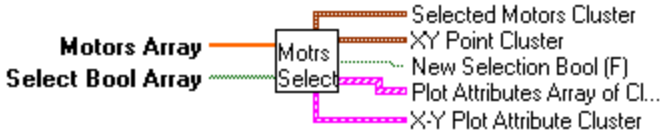


You can use the motor and CS menu rings to select motors to jog and coordinate system to convert positions to CS units. If there are motion programs running on PMAC while this VI is executing the position data plotted and displayed by this example will be from those motion programs. In either case, motor positions are displayed in real-time in the **PmacMotorsPVE** cluster and the strip chart.

To implement this exercise we use **PmacMotorJogControl** and **PmacMotorsPositions**. The CS definitions are retrieved using **PmacCoordDef** covered in a later exercise. The **PmacMotorsPVE** cluster generated by **PmacMotorsPositions** is unbundled and the array of values is passed to the **PmacMotorsPlotSelect** VI

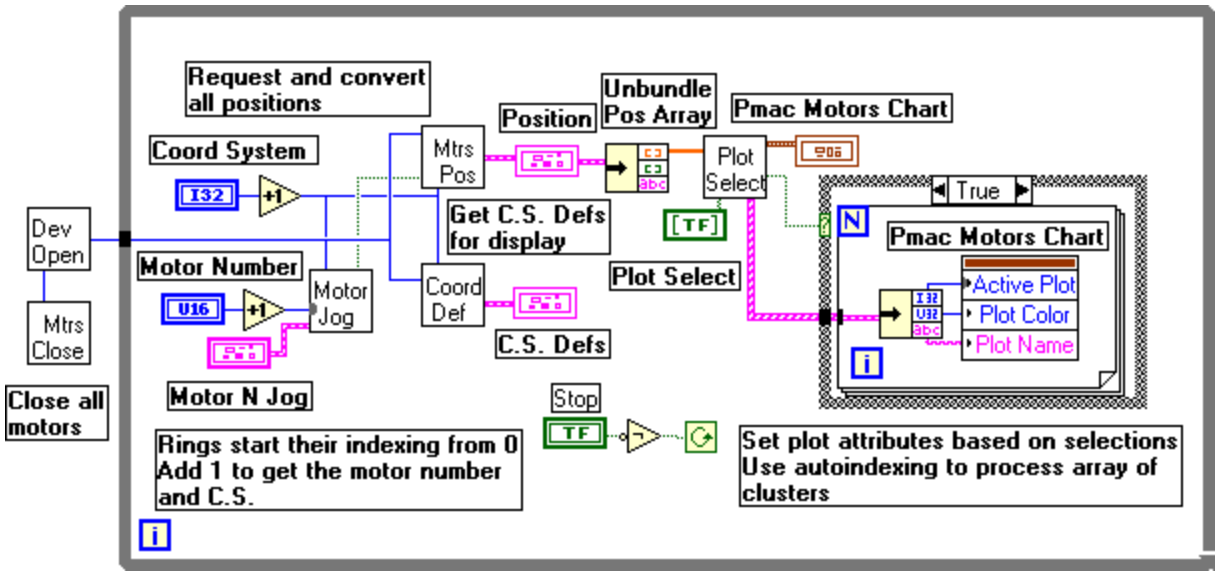
- **PmacMotorsPlotSelect** - Motors Array contains positions, velocities, or following errors for all motors on your PMAC. Select Bool Array defines which motors are copied into Selected Motors Cluster for plotting on LabVIEW strip charts. XY Point Cluster contains two values for X-Y plotting. New Selection is TRUE when Select Bool changes and indicates

the Plot Attributes Array of Clusters and X-Y Plot Attribute Cluster contain new information for updating plot legend attributes.



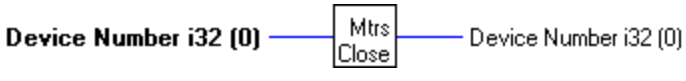
Setting the plot attributes is done in the case structure in the lower right of the diagram below. Plot Attributes Array of Clusters contains an Active Plot index, Plot Color, and Plot Name as supplied by **PmacMotorsPlotSelect**. When New Selection is TRUE the case is executed and the array is auto-indexed. The cluster of attributes is unbundled and used to set the three attributes for the plot.

One important thing to note is that Motor Number and Coord System are specified using LabVIEW rings. These controls start their indexing from 0 not 1. Hence, to use them to specify legitimate motors and coordinate systems they must be incremented by one. You can change the range on the ring from 0-7 to 1-8. However, this leaves item zero in the ring empty!

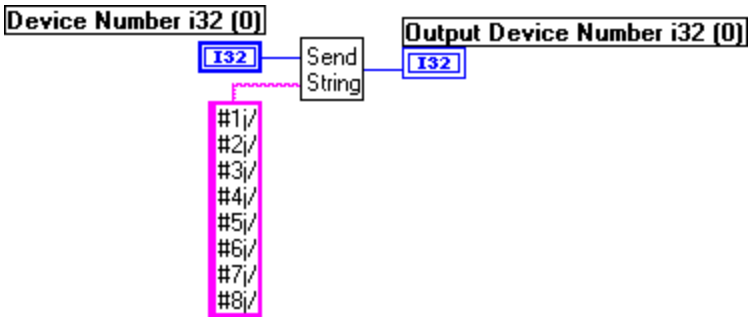


As you begin to develop your own applications, you will create your own simple VIs to do common things for you. **PmacMotorsClose** is an example of this. The VI and its diagram are shown here. If you find yourself repeatedly sending PMAC simple commands in your application you should begin creating your own set of useful VIs.

- **PmacMotorsClose** - Close all PMAC motor loops.



The important thing to note about this VI is that eight commands are executed by PMAC. A <CR> separates the individual commands in the command string.



PmacGlobal ICVs

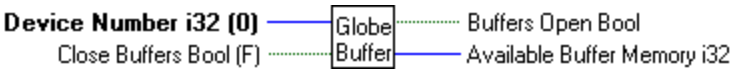
This series of exercises introduce the **PmacGlobal** collection of ICVs. These allow you to control, monitor, and configure PMAC's global characteristics.

PmacTutor11 - Configuring PMAC's Global State

This exercise introduces a number of indicators, controls, and VIs for controlling and monitoring PMAC's general operational characteristics. This is done using status indicators, a simple control cluster for sending PMAC commands, and a few clusters for configuring I-Variables. In general these capabilities are used for supervisory purposes and not exposed to general users.

The architecture for **PmacGlobal** ICVs follows those already introduced. The basics are introduced with these six VIs and their cousins.

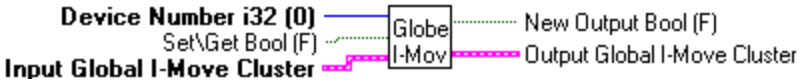
- **PmacGlobalBufferSize** - Monitor and control PMAC buffer space during system development. Buffers Open is TRUE if an open prog, open rotary, or open PLC command has been executed and the corresponding buffer has not been closed yet. Available Buffer Memory specifies how much buffer space PMAC has left for gathering and programs. A define gather command reserves all available buffer space. If Close Buffers is TRUE the gather buffer is deleted and a close command is sent to PMAC.



- **PmacGlobalControl** - Generate PMAC on-line commands for controlling PMAC program execution and save state. Command Executed Bool is TRUE when any button is clicked in Global Control Cluster.

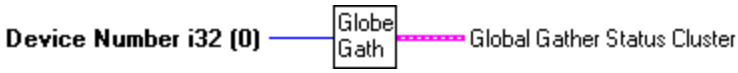
- **PmacGlobalIVarComm** - Follow PMACPanel's standard I-Variable VI architecture. When Set/Get is TRUE the global communication I-Variables are set. Otherwise they are fetched from PMAC and provided by Output Global I-Comm Cluster with New Output TRUE. Set/Get is not required and defaults to a Get operation.

- **PmacGlobalIVarMove** - Follow PMACPanel's standard I-Variable VI architecture. When Set/Get is TRUE the global movement I-Variables are set. Otherwise they are fetched from PMAC and provided by Output Global I-Move Cluster with New Output TRUE. Set/Get is not required and defaults to a Get operation.

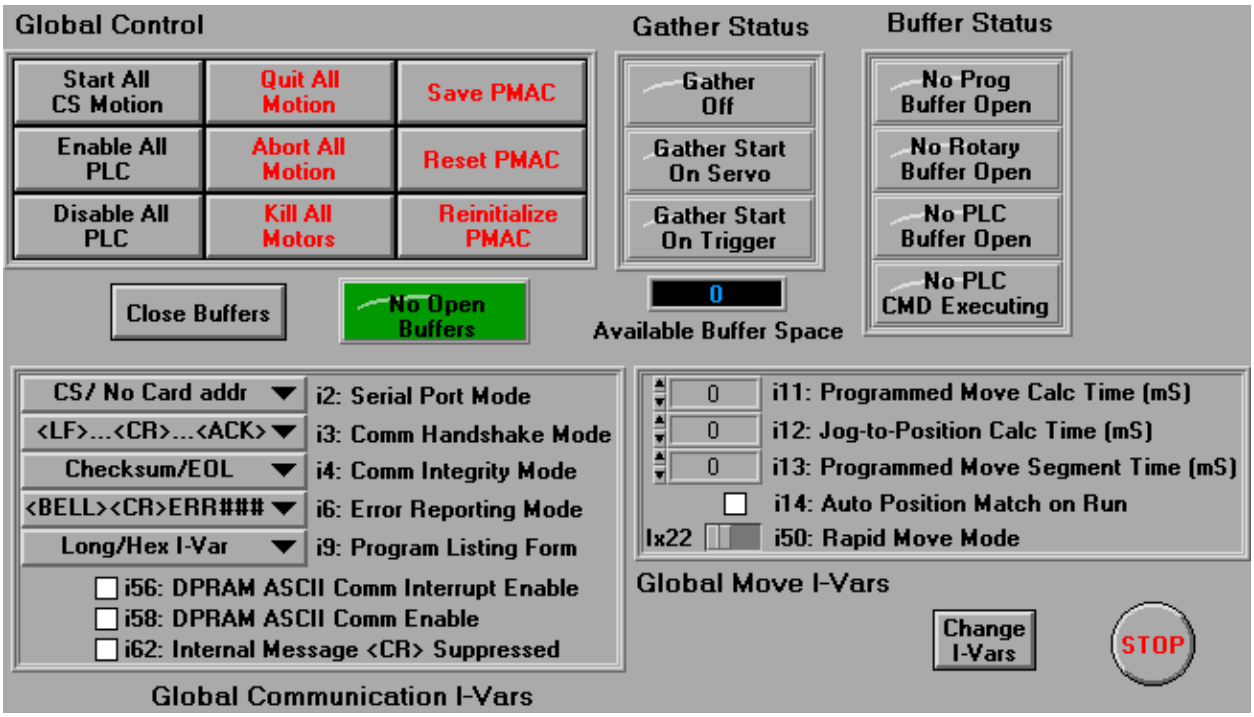


- **PmacGlobalStatBuffer** - Create a status indicator cluster for the PmacGlobalStatBuffer indicator containing PMAC's global buffer status.

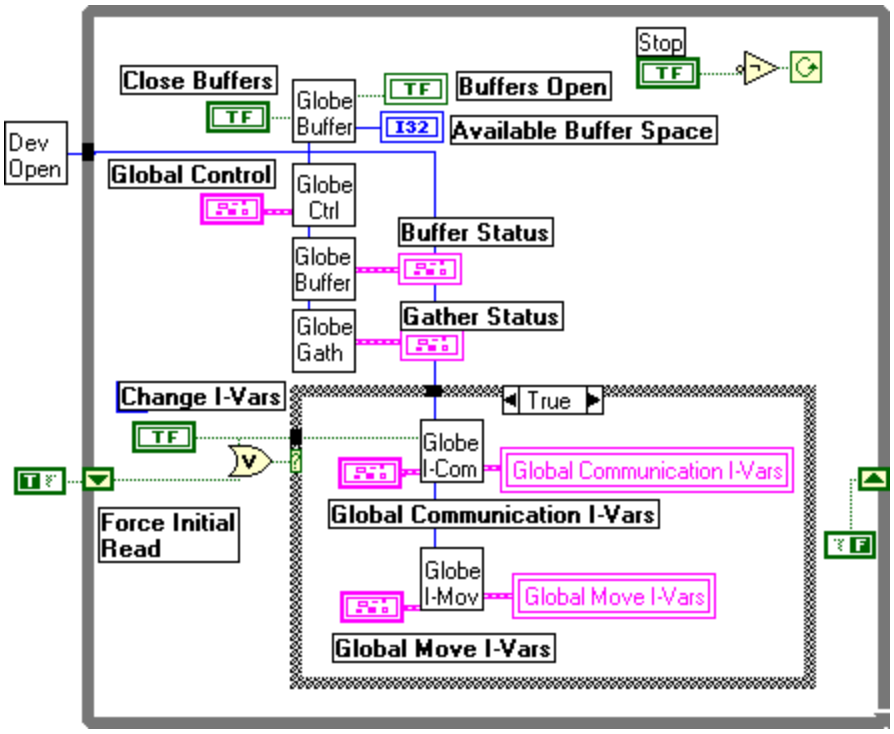
- **PmacGlobalStatGather** - Create a status indicator cluster for the PmacGlobalStatGather indicator containing PMAC's global gather status.



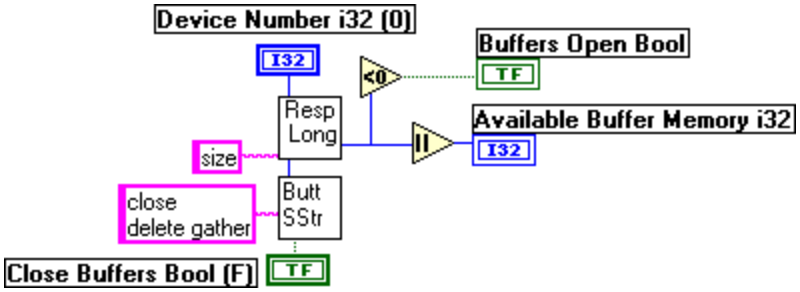
The panel for this exercise is the basis for one of PMACPanel's terminal tools. At the top is the **PmacGlobalControl** cluster. It enables you to generate global commands for PLC and program control. During your development, you might want to Abort All Motion. The Save, Reset, and Reinitialize buttons allow you to save PMAC's configuration to its onboard battery backed RAM or flash memory. This way PMAC will always boot with the proper program and configuration information for your application. The two status indicator clusters might be useful during development as will the ability to monitor PMAC's available buffer space. The two I-Variable clusters are useful when you are configuring communication or require specific program execution characteristics. For example, later exercises execute multi-axis motion using circular interpolation. This requires I13 > 0.



The diagram for this exercise should begin looking familiar. Most of the work is contained in the provided PMACPanel ICVs. The control cluster provides the input for **PmacGlobalControl.** Status VIs process requests for PMAC global status and create appropriate clusters. The architecture for configuring the I-Variables is slightly different from that used to configure motors. There is no Motor Number equivalent that can change. Instead, a shift register is initially set to TRUE to force an I-Variable read and set FALSE for every other iteration.



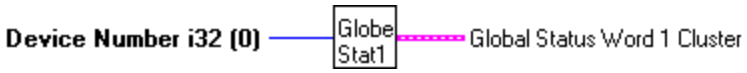
PmacGlobalBufferSize provides two capabilities. The VI queries PMAC for available buffer size and parses the state of the buffers and their size. When Close Buffers is TRUE PMAC executes a command to close the open buffers and delete the gather buffer. Generally, an open gather buffer prevents the download of any new program or modification of any existing program.

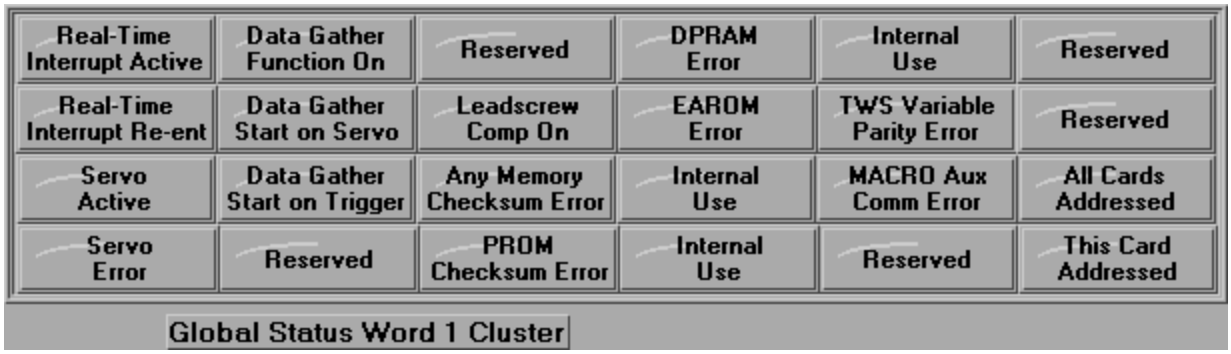


Global Status ICVs

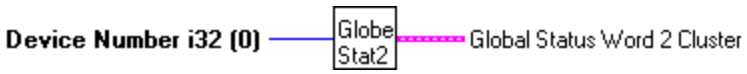
In typical fashion, the PMACPanel global collection of ICVs provides a pair of indicator clusters and associated VIs to monitor all global status bits. The icons for the VIs and the indicator clusters are shown below. Unlike the **PmacMotor** status word clusters these clusters have many status bits that are reserved and for internal use.

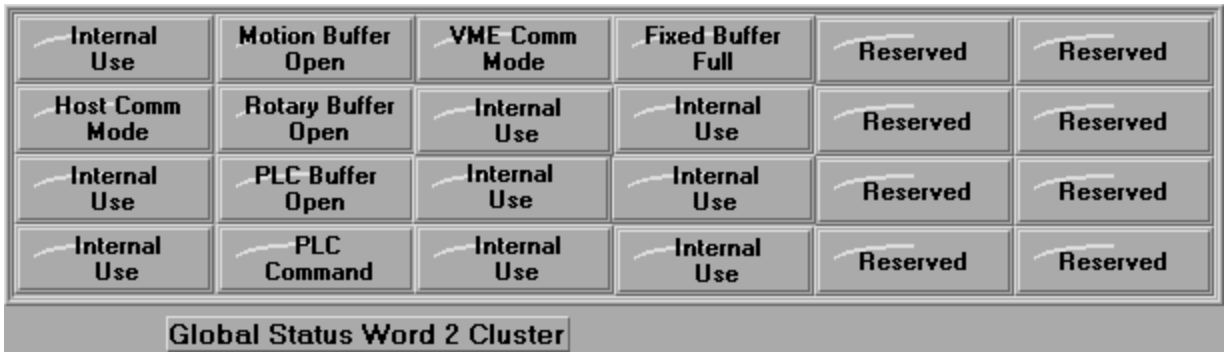
- **PmacGlobalStatWord1** - Create a status indicator cluster for the PmacGlobalStat1 indicator containing PMAC's global status.





- **PmacGlobalStatWord2** - Create a status indicator cluster for the PmacGlobalStat2 indicator containing PMAC's global status.





PmacCoord ICVs

This series of exercises introduce the **PmacCoord** collection of ICVs. Coordinate systems organize motors into familiar engineering measurement systems in which motion programs execute. They define the scaling between motor encoder counts and engineering units such as inches, centimeters, degrees, or radians. They can also define coupling between multiple motors and a single coordinate axis. You will not use these ICVs to configure your coordinate systems. But you can use them to query the coordinate system configuration so that motor motion can be specified in coordinate system units rather than encoder counts. Generally, you will not work with these VIs at this level. Their capabilities are integrated into other collections of VIs.

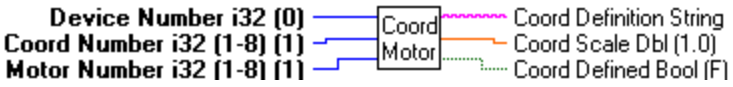
PmacTutor12 - Using Coordinate System Definitions

This exercise introduces ICVs for determining coordinate system definitions and parsing these definitions into LabVIEW data types that can be used to convert between encoder counts and coordinate system units and determining which motors, if any, are defined in a given coordinate system.

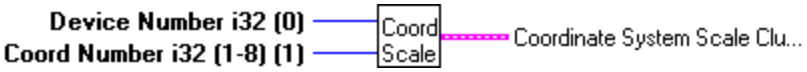
The architecture for the following **PmacCoord** ICVs is a little different from that used for status and other fundamental queries. A few of these VIs make use of local state variables to keep track of queries for coordinate system definitions so that these queries are not placed every time the VI is executed within your VI. This reduces communication traffic and relieves the developer from having to check for a new query. The fundamental assumption in this approach is that after you create your application you will not constantly redefine the specified coordinate system. When you've requested a coordinate system definition from PMAC and change it, the new definition will not be reflected in your application. To refresh the definition, temporarily request the definition for another coordinate number or close your VI and reopen it.

The architecture for determining coordinate system definitions relies on the following three VIs. They perform similar operations but return coordinate system definitions for different purposes.

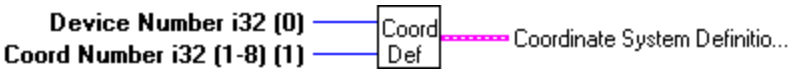
- **PmacCoordMotorDef** - Query PMAC for the definition of Motor Number in Coord Number. If the motor is not defined in the specified CS Coord Definition is "Encoder", Coord Scale = 1.0, and Coord Defined is FALSE. If the motor is defined in the specified CS Coord Defined is TRUE, Coord Scale is the encoder to CS unit scale factor, and Coord Definition is the definition (e.g. "#1->16000X").



- **PmacCoordScale** - Query PMAC for the motors defined in Coord Number. The Coordinate System Scale Cluster (PmacCoordScale.ctl) contains three arrays with the motor definition, scale factor, and whether or not the motor is defined in Coord Number. The actual query is only placed if Coord Number changes from a previous call.



- **PmacCoordDef** - Fetch the motor scaling definitions for the specified coordinate system and provide a cluster for the PmacCoordDef indicator.



There are a few limitations you should be aware of when querying coordinate system definitions from PMACPanel. A motor is generally assigned to a single coordinate axis as in the following definition

```
&1
#1->1000X
```

This specifies motor #1 as belonging to coordinate system &1 and that 1 X unit is 1000 encoder counts. The scale factor would thus be 1000.

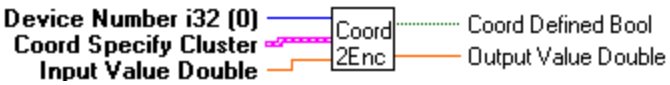
The limitation arises when coordinate system axes are linear combinations of several motors as in this example that rotates the coordinate system 30 degrees from the mechanical axes of the motor

```
&1
#1->8660.25X-5000Y
#2->5000X+8660.25Y
```

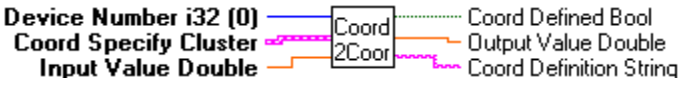
There is no easy way to parse this information when returned by PMAC and present it to you for use because there are so many possible ways of defining coupled motor systems. Furthermore, the individual items in the **PmacMotors** indicator clusters as defined would change definition every time you switch from encoder counts to CS units. If your axes are coupled like this you should study the VIs presented here and modify them for your own use or you can hard code the scaling and motor state conversions into your application.

There are three VIs that use the scale factors provided by the previous VIs to convert numerical inputs from encoder counts to coordinate units and back. This can be done for individual motors or all motors in a coordinate system.

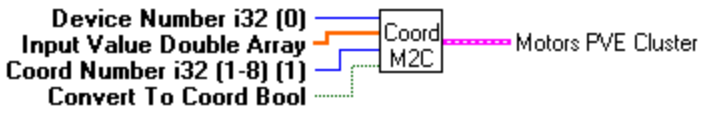
- **PmacCoordMotorToEncoder** - Coord Specify Cluster specifies a motor within a CS and an attempt to convert Input Value from CS units to encoder counts. If the motor is not defined in the CS no conversions is applied. If the motor is defined and Convert is TRUE Coord Defined is TRUE and Output Value is scaled from CS units to encoder counts.



- **PmacCoordMotorToCoord** - Coord Specify Cluster specifies a motor within a CS and an attempt to convert Input Value from encoder counts to CS units. If the motor is not defined in the CS no conversion is applied. If the motor is defined and Convert is TRUE Coord Defined is TRUE and Output Value is scaled from encoder counts to CS units. Coord Definition is a string specifying Output Value units as "Encoder" or the CS definition of the motor.

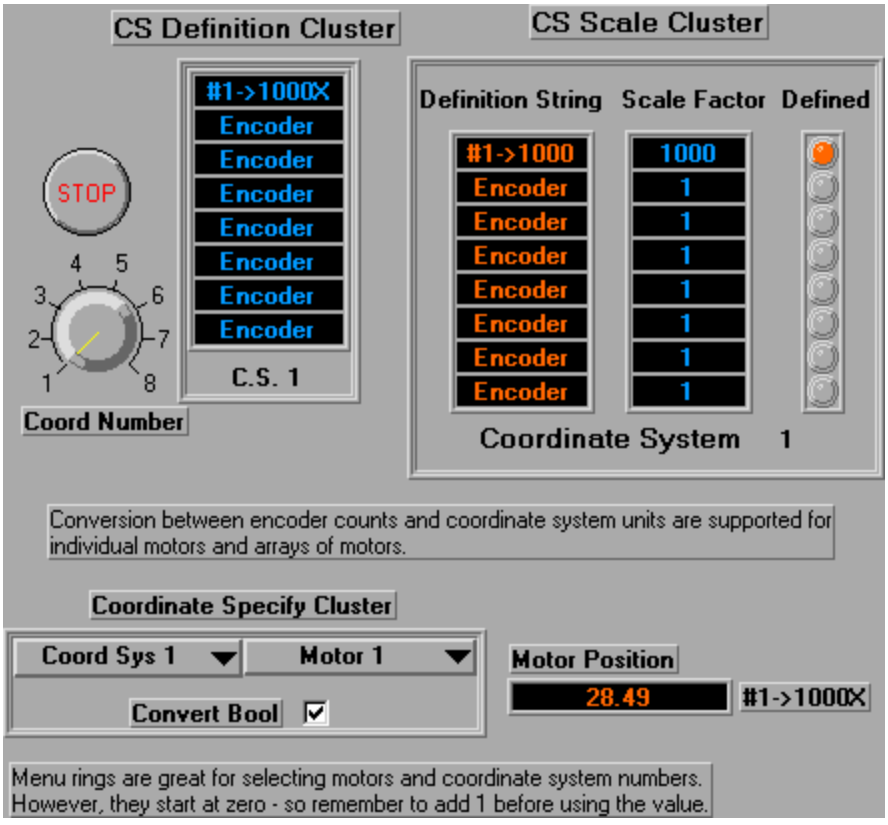


- **PmacCoordMotorsToCoord** - Generate an indicator cluster for PmacMotorsPVE.ctl. Input Value Double is an array of positions, velocities, or following errors from VIs in the PmacMotors collection. If Convert To Coord is TRUE fetch the CS definitions for the motors specified in Coord Number and scale them to CS units. Motors not defined in Coord Number are not scaled.



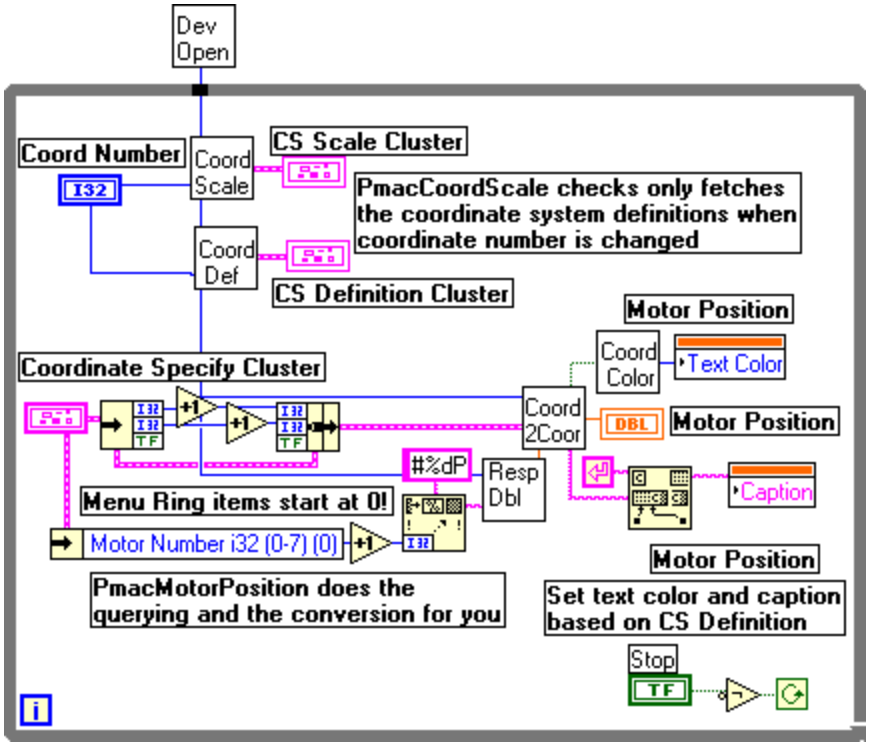
The panel shows two indicator clusters. The CS Scale Cluster contains the definition of all motors in the specified coordinate system as a displayable string, a numeric scale factor, and a Boolean indicating whether that motor is defined in the coordinate system. In the example, motor #1 is defined in coordinate system 1. The orange text color indicates that the CS defined in CS Scale Cluster's caption is being addressed. The CS Definition Cluster is a derivative of the larger cluster and can be used in conjunction with **PmacMotorsPVE** on your application panels.

The lower portion of the panel is a modified **PmacCoordSpecify** cluster used to specify a motor, coordinate system, and conversion from encoder counts to coordinate system units. The modifications were made by replacing individual control items in the stock cluster with types that are more appropriate. The Orange numeric position indicator and its caption indicate the motor definition within the specified Coord Number.



The diagram for the query of coordinate system definitions is simple. The lower portion of the diagram demonstrates how to use the conversion VIs to convert and display the motor position data. The Coordinate Specify Cluster in this example is made from Menu Rings whose index always starts at zero. Because PMAC motors and coordinate systems start their number at one you must add one to the selection index. This is not necessary if you use numeric controls in your Coordinate Specify Cluster.

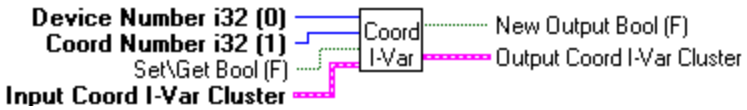
The motor position is processed by **PmacCoordMotorToCoord** to produce three outputs that can be used to enhance the display of the data. **PmacCoordColor** sets the color of the numeric indicators. The Coordinate Definition String is used to set the indicator's caption after stripping the terminating <CR>. The example shows the use of both named and unnamed unbundles to get the data required for the operation. Again, you will most likely not work with these VIs at this level.



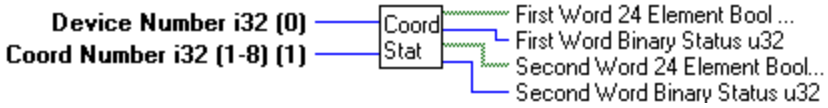
PmacTutor13 - Configuring and Monitoring Coordinate Systems

This exercise introduces ICVs for monitoring and configuring coordinate systems and program execution within coordinate systems. These VIs follow the same I-Variable and status architectures already introduced. The VI's are

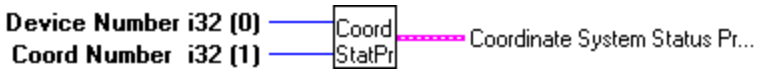
- **PmacCoordIVar** - Follow PMACPanel's standard I-Variable VI architecture. When Set/Get is TRUE the I-Variables for the specified Coord Number are set. Otherwise they are fetched from PMAC and provided by Output Coord I-Var Cluster with New Output TRUE. Set/Get is not required and defaults to a Get operation.



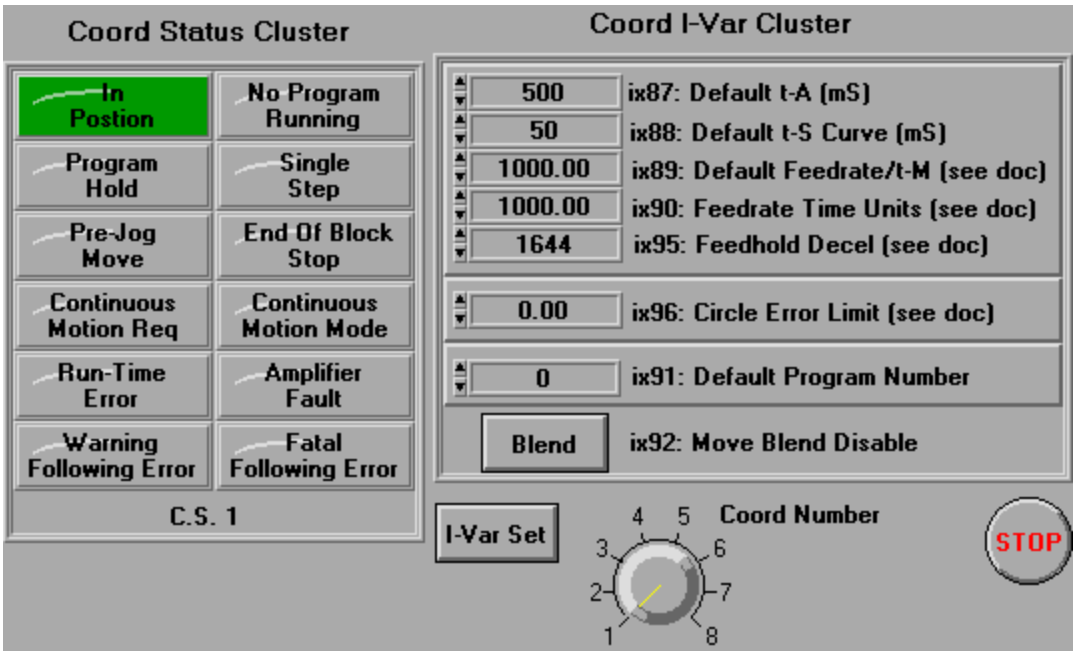
- **PmacCoordStat** - Query PMAC for the status of the CS specified by Coord Number. Report the two status words as arrays of Booleans and unsigned 32 bit integers.


Device Number i32 (0)
Coord Number i32 (1-8) (1)
Coord Stat
First Word 24 Element Bool ...
First Word Binary Status u32
Second Word 24 Element Bool...
Second Word Binary Status u32

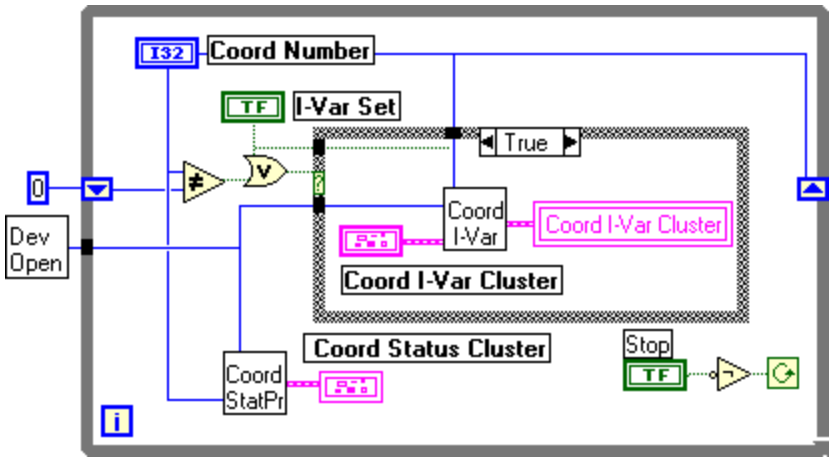
- **PmacCoordStatProg** - Create a status indicator cluster for the PmacCoordStatProg indicator containing the status for Coord Number.



Monitoring the coordinate system status is a very common operation because programs run in coordinate systems. If your coordinate system has Cartesian X-Y-Z axes and rotary U-V-W axes then your program executes its moves using the motors assigned to those axes. The coordinate system status reflects the state of the executing program and the combined state of all motors in the system. If all motors are in-position then the coordinate system is in position. If any motor has a warning following error then the coordinate system has a warning error.



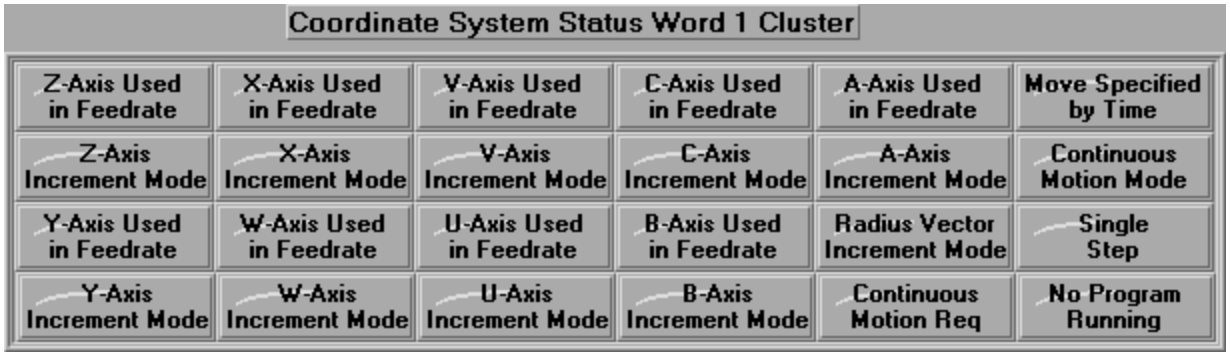
The diagram also has a familiar architecture.



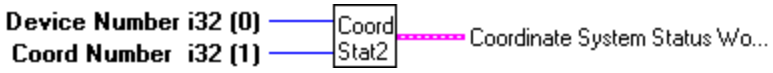
Coordinate System Status ICVs

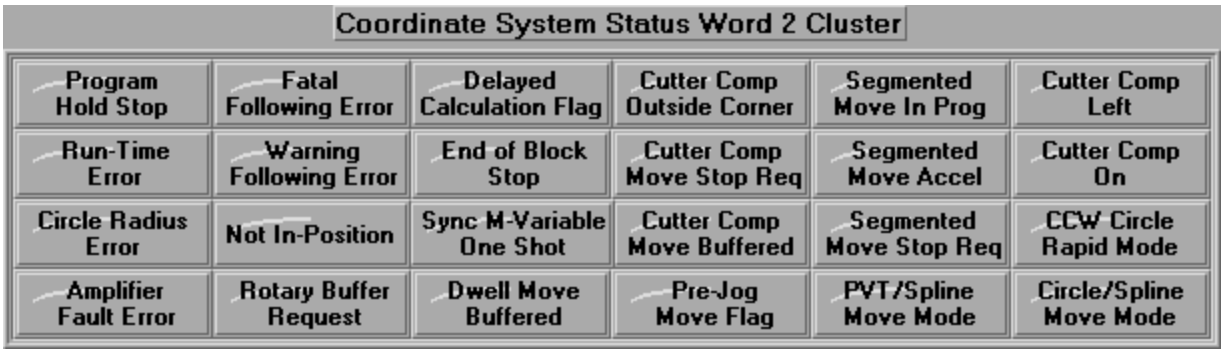
There are, as expected, indicators for both coordinate system status words and a few miscellaneous VIs that will be introduced in later exercises.

- **PmacCoordStat1** - Create a status indicator cluster for the PmacCoordStat1 indicator containing the status for Coord Number.



- **PmacCoordStat2** - Create a status indicator cluster for the PmacCoordStat2 indicator containing the status for Coord Number.





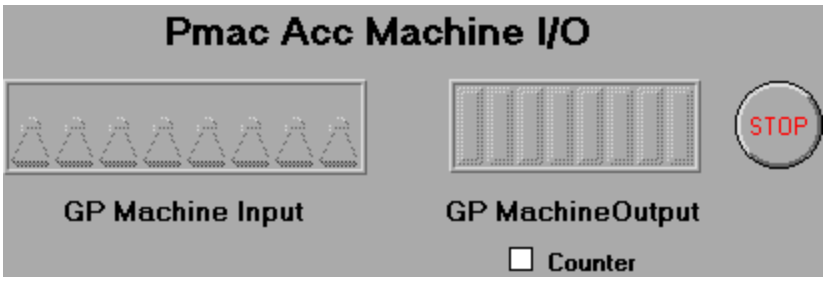
PmacAcc ICVs

This series of exercises introduce the **PmacAcc** collection of ICVs. These form templates that combine **PmacMemory's** direct access of memory with **PmacIVar's** Set/Get architecture that access PMAC's memory mapped devices. Using this approach, you can hide the bit field and address specifications in your VIs.

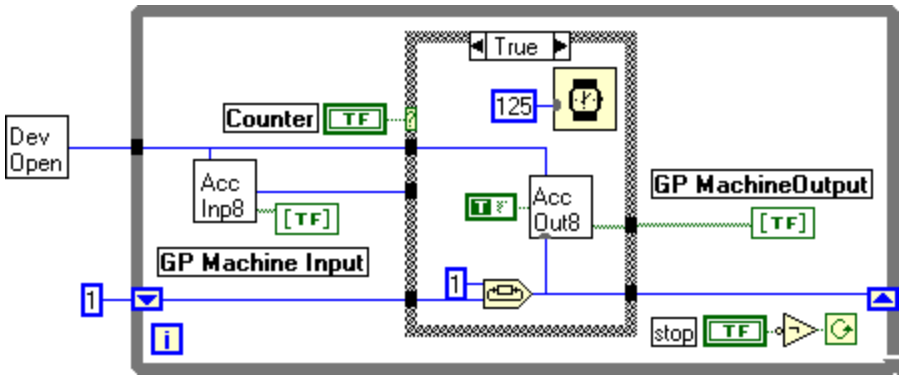
This collection of VIs will grow as Delta Tau adds support for its numerous accessories. At present, we will demonstrate the Machine Input/Output VIs and a simple example of the ACC16D control panel.

PmacTutor14 – Machine Input and Output

This tutorial demonstrates two VIs. One that allows you to access the general-purpose machine input port and one that accesses the output port. On the demonstration box used for the development of PMACPanel, switches drive the inputs and the outputs drive a set of LEDs. When running the example, the GP Machine Input indicator directly reflects the operation of the physical switches.

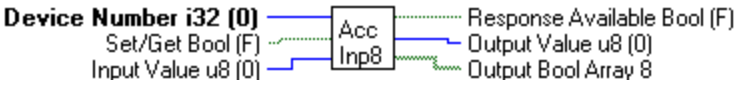


The physical inputs are copied into the output port located in a different bit field to drive the GP Machine Output array and the LEDs on the physical device. If you check the Counter box a bit is rotated through a number and written the physical port thereby driving the LEDs. This is shown on the diagram below.

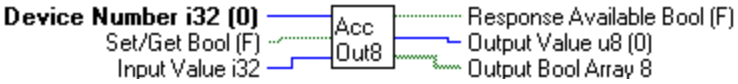


The two VIs that implement the example require a little discussion so that you can comfortably develop or modify your own **PmacAcc** VIs.

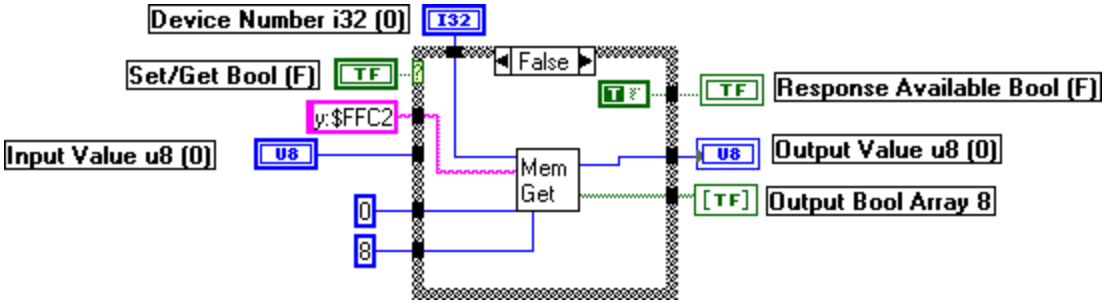
- **PmacAccMachineInput8** - If Set\Get is FALSE or not wired get the Machine Input Port contents. Response Available will be TRUE to indicate the Outputs contain the value. If Set\Get is TRUE set the Machine Input Port using Input Value. Response Available will be FALSE and the outputs default to Input Value.



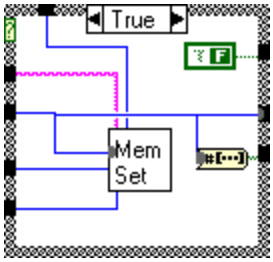
- **PmacAccMachineOutput8** - If Set\Get is FALSE or not wired get the Machine Output Port contents. Response Available will be TRUE to indicate the Outputs contain the value. If Set\Get is TRUE set the Machine Output Port using Input Value. Response Available will be FALSE and the outputs default to Input Value.



The diagram for **PmacAccMachineInput8** is shown here to make a point about the **PmacMemory** VIs. The VI fetches the machine input port data located at Y:$FFC2 bits 0-8 using **PmacMemoryGet** – simple enough. The issue you need to watch involves the Set case using **PmacMemorySet**.

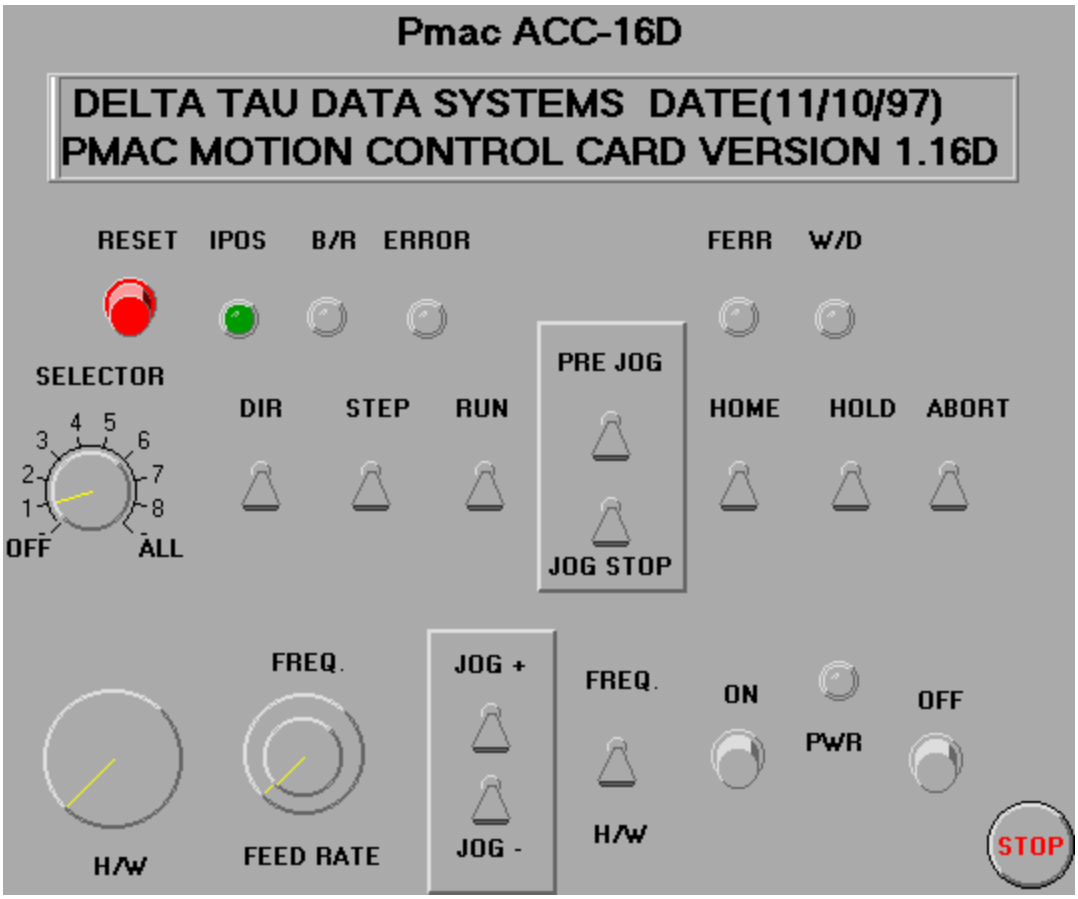


This case illustrates an important behavior associated with **PmacMemorySet**. This VI sets the contents of the bit field and provides as its output the entire 24-bit word. This was done so that multiple copies of the VI can be chained together to handle multiple bit fields. If you take the output of **PmacMemorySet** and wire this to Output Value for **PmacAccMachineInput8** Output Value will not be what you expect. This is obvious when running **PmacAccMachineOutput8**. To remedy this you need to use the Input Value that is used to set the field as Output Value.

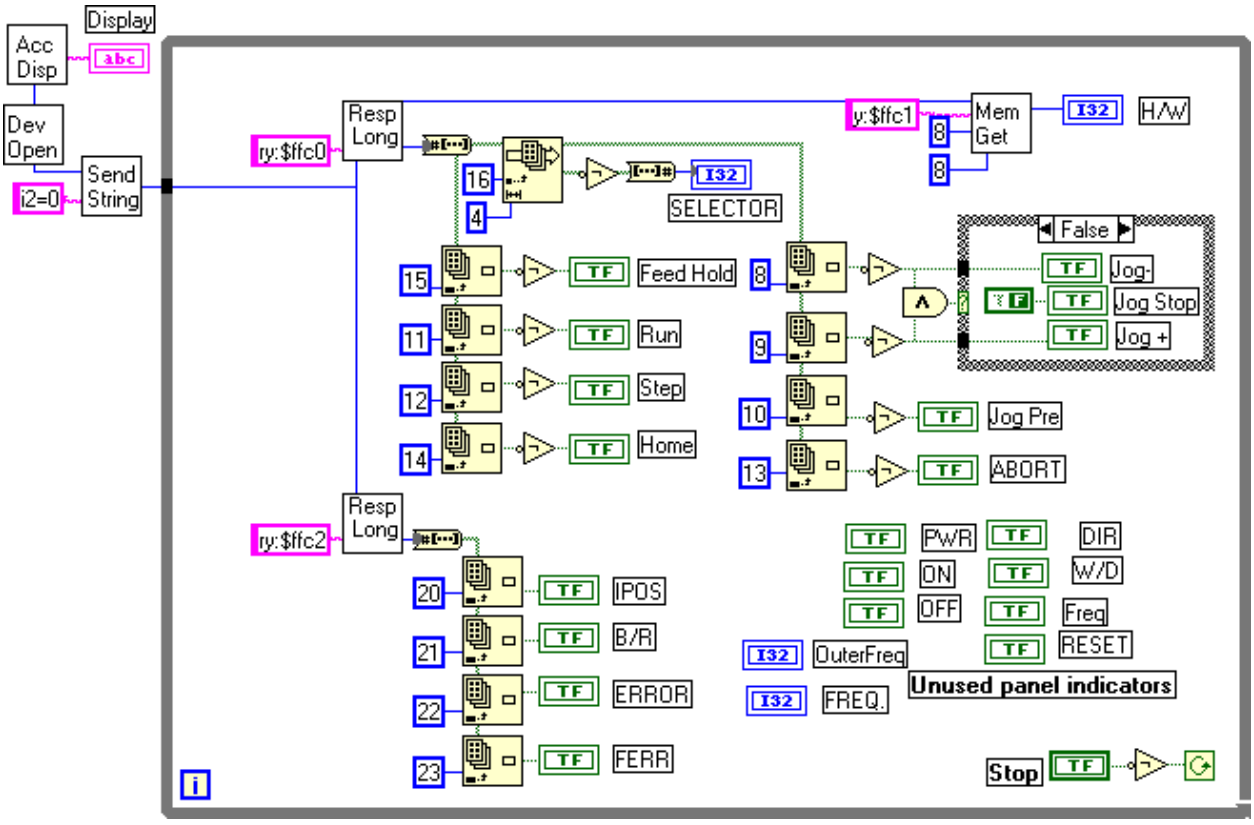


PmacTutor15 – ACC16D Control Panel

This example fetches the contents of the several memory locations for the Control Panel port at Y:$FFC0, Y:$FFC1, and Y:$FFC2. These registers allow the ACC16D accessory to control the operation of motors and programs from an operator control panel. The panel for the example responds to the physical panel by mimicking the switch operation. If you have an ACC16D and run this example you will see the various switches on this panel respond in kind.



The diagram shows that PMAC is queried for the contents of the three locations and the converted into appropriate types for processing. The selector field is extracted, as are the individual control/status bits. If you want to make this panel control PMAC you can use the same control layout and generate the appropriate commands by borrowing portions of **PmacMotorJog**, etc. We have not done this here.


Acc Disp
Display
Dev Open
Send String
i2=0
Resp Long
ry:$ffc0
16
4
I32
SELECTOR
15
Feed Hold
11
Run
12
Step
14
Home
Resp Long
ry:$ffc2
20
IPOS
21
B/R
22
ERROR
23
FERR
y:$ffc1
Mem Get
8
8
I32
H/W
8
9
10
Jog Pre
13
ABORT
False
Jog-
Jog Stop
Jog +
PWR
DIR
ON
W/D
OFF
Freq
RESET
I32
OuterFreq
I32
FREQ.
Unused panel indicators
Stop
i

Chapter 5 - Development Tools

Basics

Serious PMAC configuration, tuning, and setup requires the use of Pewin32. Once this step is completed, development of your PMACPanel application can begin. PMACPanel supplies a number of tools and application VIs to aid in this process and provides an architecture for adding more.

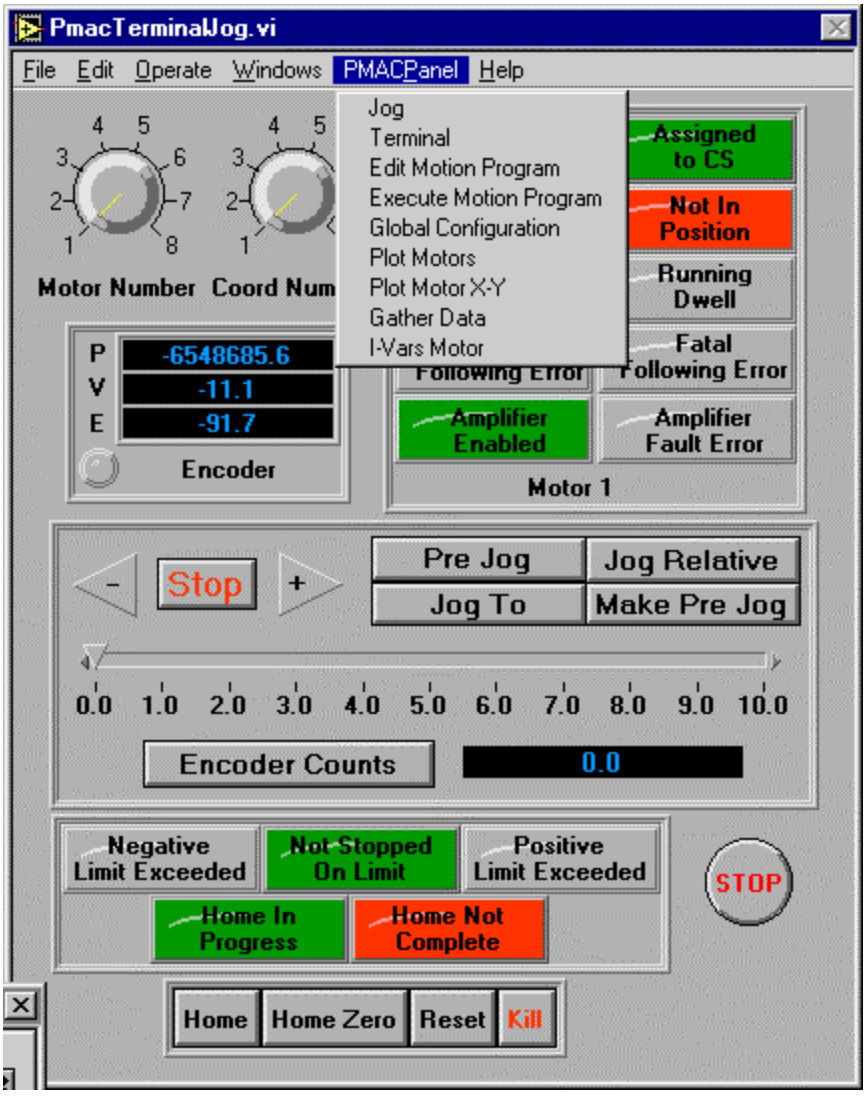
There are 10 standalone development tools covered in this chapter. They make extensive use of the ICVs introduced in Chapter 4 and form an excellent introduction to the PMACPanel integration ICVs introduced later in Chapter 6.

- **PmacTerminal** - A basic ASCII terminal with useful ICVs for monitoring coordinate system and motor status. In addition, several new ICVs for controlling programs and PLCs are introduced.
- **PmacTerminalEdit** - A simple editor for program development and downloading. The tool also supports the creation of encapsulated motion and PLC program VI wrappers that hide the details involved in controlling and monitoring PMAC motion and PLC programs.
- **PmacTerminalExecute** - An interactive debugger for monitoring and controlling the execution of PMAC programs.
- **PmacTerminalJog** - A simple tool for jogging and controlling motors
- **PmacTerminalMotors** - A graphical tool for monitoring and plotting the motion of multiple motors in a real-time strip chart.
- **PmacTerminalMotorsX-Y** - A tool for monitoring and plotting the motion of motors in a real-time X-Y chart.
- **PmacTerminalGather** - A tool for specifying and gathering PMAC motion data and exporting it to Microsoft Excel.
- **PmacTerminalMotorIVars** - A tool for configuring individual motor I-Variables.
- **PmacTerminalCoordIVars** - A tool for configuring coordinate system I-Variables and monitoring coordinate systems

- **PmacTerminalGlobal** - A tool for monitoring PMAC's state, saving configurations, and configuring important global I-Variables.

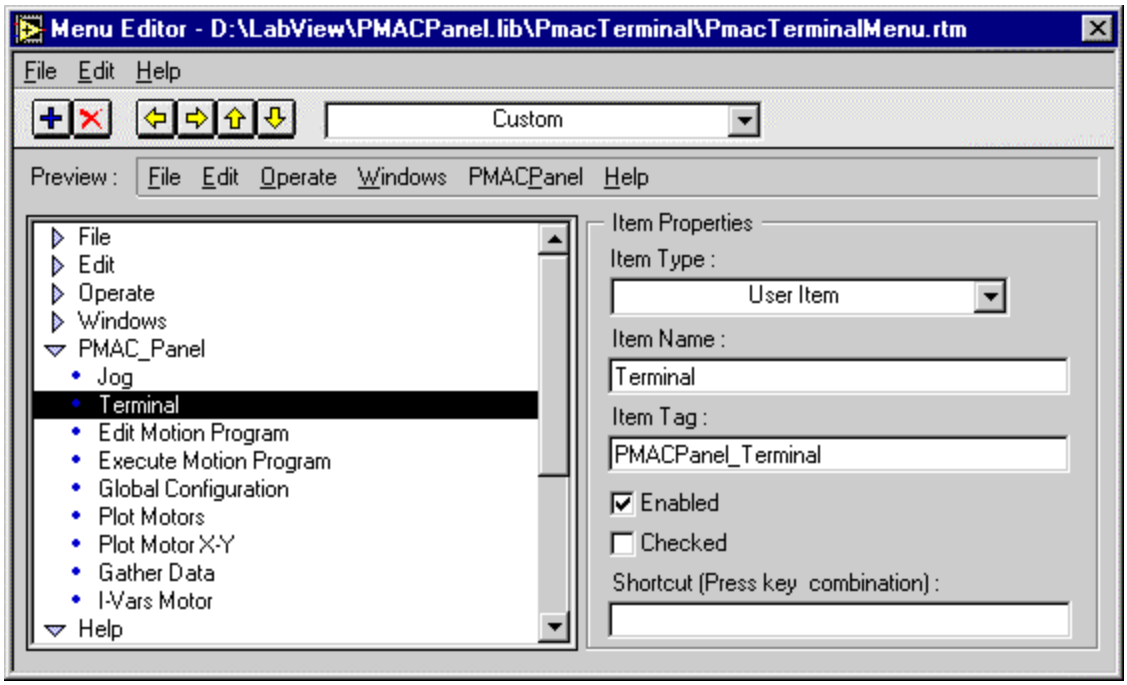
Tool Menus

The tools distributed with PMACPanel can be integrated into your own application using LabVIEW 5.0's new ability to edit and process menubar selections. This allows your applications easy access to the development tools and an easy way to add your own custom tools. The figure below shows the panel for the **PmacTerminalJog** tool. The menu bar for the tool shows an entry for PMACPanel that contains the menu items for the development tools.



Modifying the menu

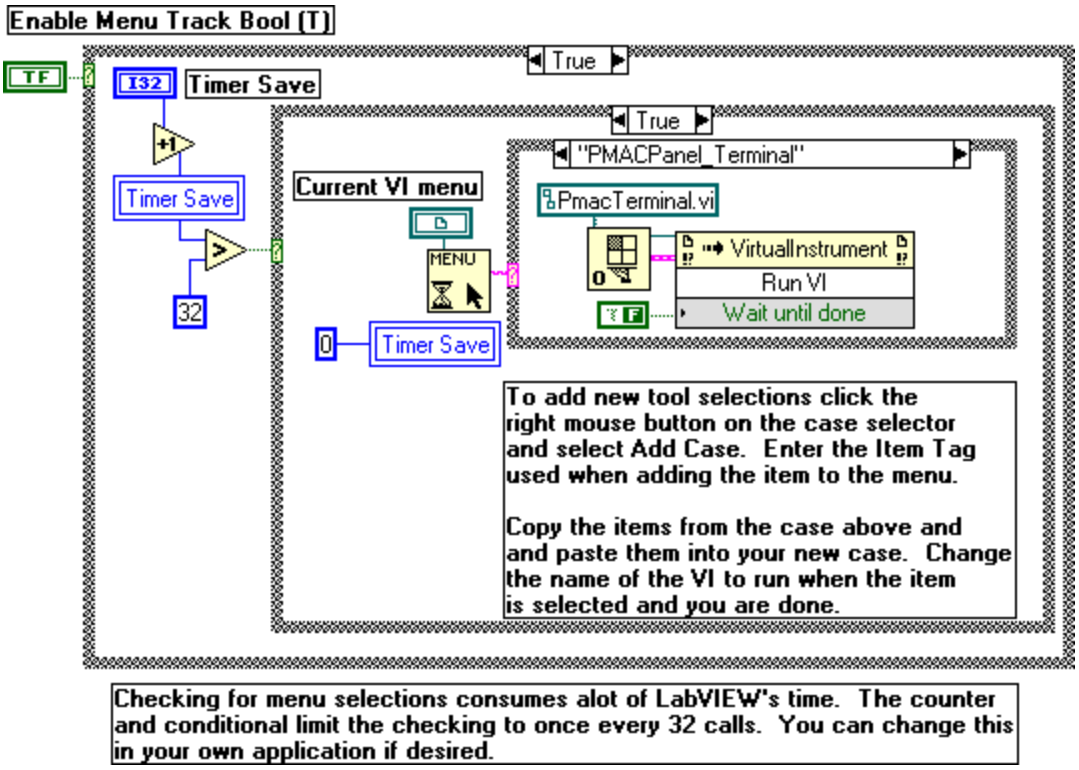
This is a custom menu named **PmacTerminalMenu.rtm** that is set as the default for your application VIs run-time menu. To modify your application VI's menu or modify the existing one use the **Edit»Edit Menu** option from your VIs panel to display to following dialog.



Select **File»Open** to display the selection box then double click on **\PmacTerminal\PmacTerminalMenu.rtm** or click OK. This will change your VIs run-time menu from the default or minimal options to custom. You will note that the panel shows an Item Name as it appears in the menu and an Item Tag that is used by **PmacTerminalMenu.vi** to decode your selection and execute the desired tool. To add your own tools to the menu highlight an existing menu item entry and select **Edit»Insert User Item**. Give the new entry a name and a tag. Save the modifications. All existing PMACPanel tools using the menu capabilities will now reflect your changes. If you've added a new tool, you must make some additions for your new VI.

Modifying PmacTerminalMenu

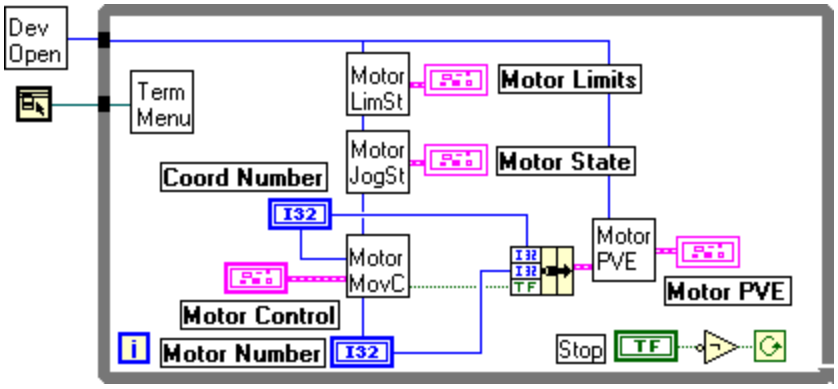
To actually implement the new selection you need to add it to **PmacTerminalMenu**. The diagram for the VI is shown here. The instructions direct you to add a case for the Item Tag specified in the Edit Menu panel and to copy the VIs to execute your VI into the new case. Once completed all tools can access each other without needing any new configuration.


Enable Menu Track Bool (T)
TF
I32
Timer Save
True
+1
Timer Save
>
32
True
"PMACPanel_Terminal"
Current VI menu
PmacTerminal.vi
MENU
VirtualInstrument
Run VI
Wait until done
0
Timer Save
To add new tool selections click the right mouse button on the case selector and select Add Case. Enter the Item Tag used when adding the item to the menu.
Copy the items from the case above and and paste them into your new case. Change the name of the VI to run when the item is selected and you are done.
Checking for menu selections consumes alot of LabVIEW's time. The counter and conditional limit the checking to once every 32 calls. You can change this in your own application if desired.

There are a few limitations associated with using LabVIEW's menubar capabilities. The main one is that it noticeably slows applications that are doing highspeed query/response updates of status and position indicators. To remedy this the **PmacTerminalMenu** VI is configured to execute the actual checking of the menu selection once every 32 calls. You can change this logic or add timers to suit your own needs.

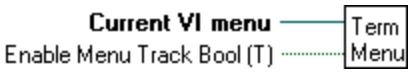
Basic Tool VI Requirements

PMACPanel's use of the menubar selection VIs spawns the VI specified by the selection as a separate concurrently executing task. This means the VI should be organized much like the exercises. Open PMAC, run a while loop, use s Stop button. The diagram for **PmacTerminalJog** is show here to illustrate this.



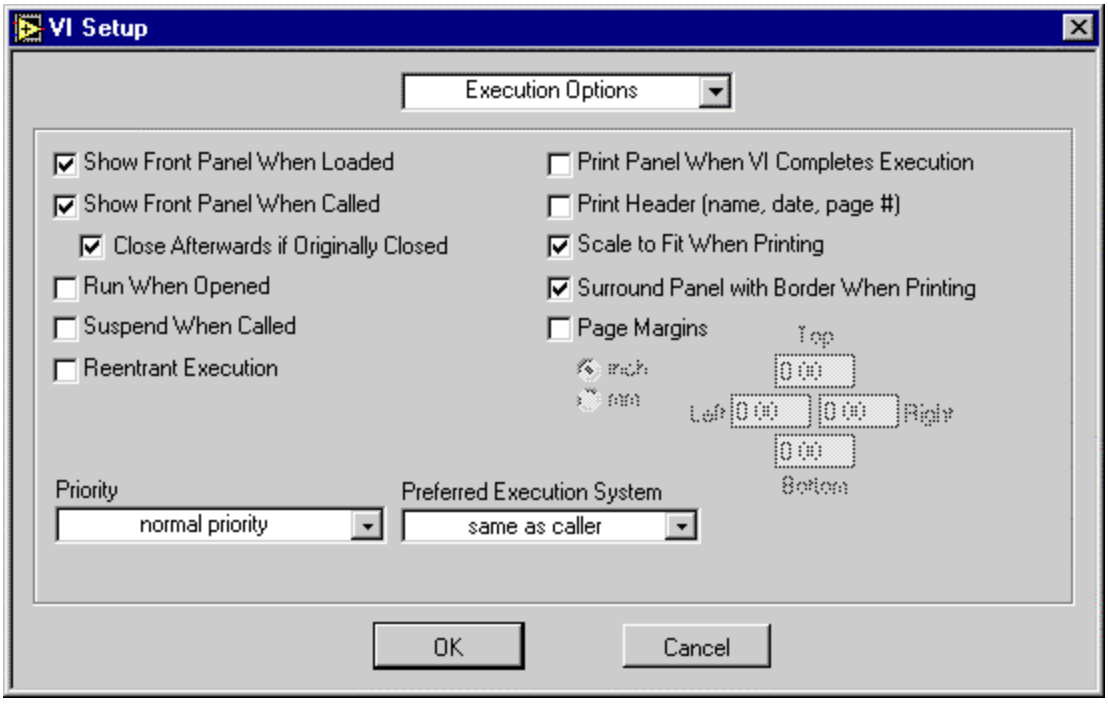
The only modification is the addition of the **PmacTerminalMenu** VI that actually spawns the new VI in response to a menu selection and the Current VIs Menu function selected for the function palette.

- **PmacTerminalMenu** - This VI uses the Current VI Menu supplied to it to check for selections. Because you may have time critical operations where you don't want to check for selections, you can set Enable Menu Track FALSE using a switch to disable the testing. This is done in the PmacTerminalMotors tool because the delay causes a noticeable blip in plotted position data.

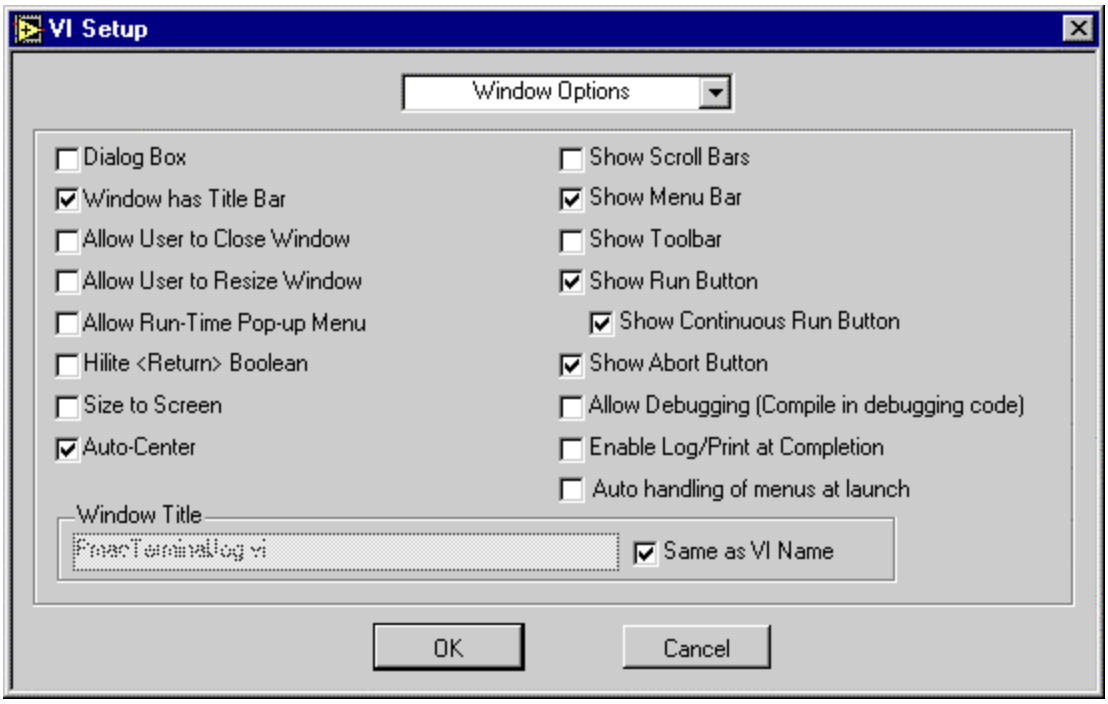


Basic Tool VI Configuration

When a tool is spawned the Execution and Window options you define for the VI Setup are important. The following panel shows the Execution Options used for a PMACPanel tool. The panel should be shown when loaded otherwise selecting the tool runs it without a panel - not very useful. If you call the VI instead of spawn it then you want the Show Front Panel When Called and then closed afterward. The problem with calling the VI rather than spawning it is that it must complete before the caller's panel becomes active again. Not a flexible system but it maybe what you want.

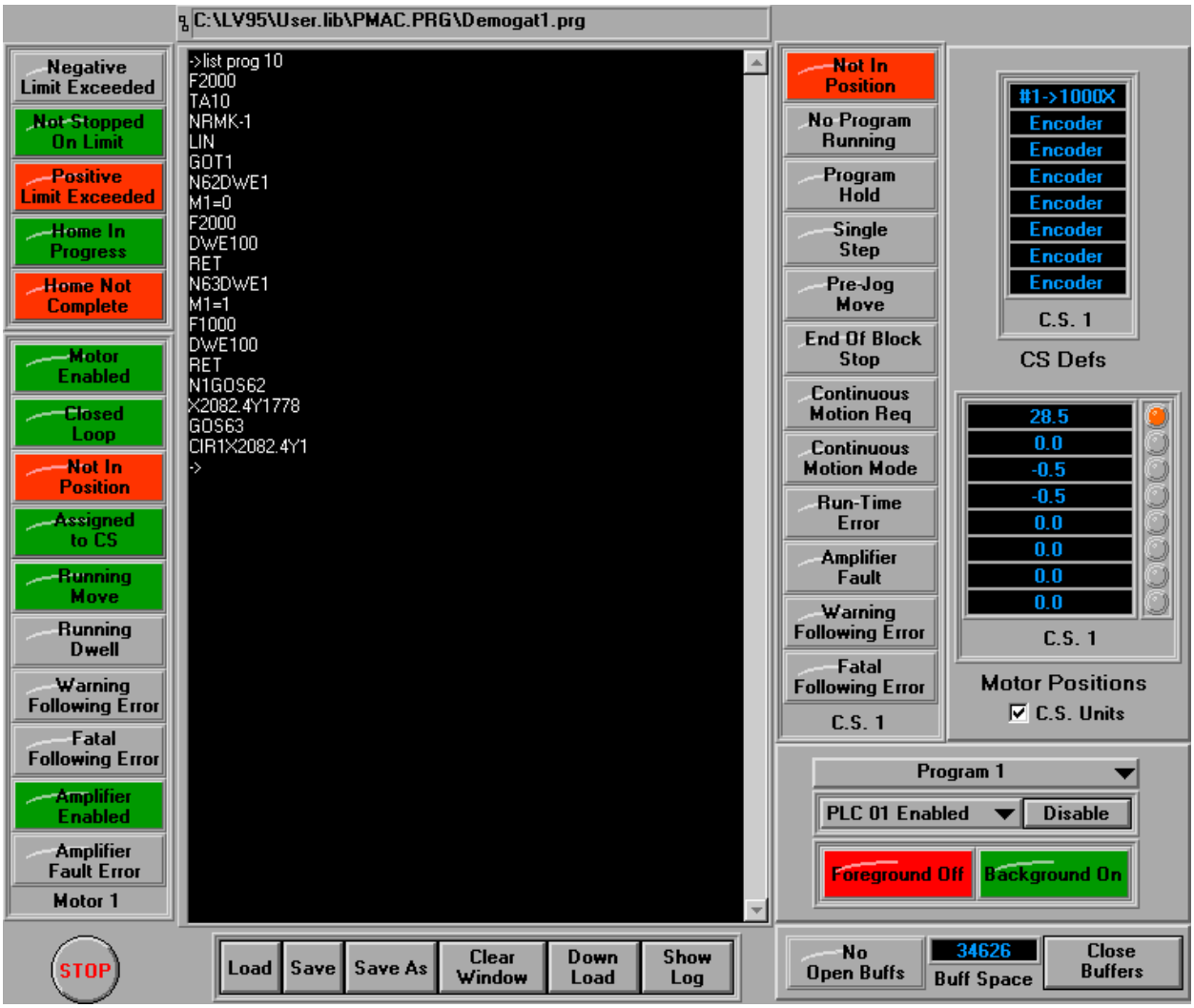


Generally, the panel window should prevent users from doing too many things. PMACPanel tools have a title bar, and auto-center. When in edit mode the menu bar, run button, and abort button are shown. When running these disappear and the user must use the menubar of the window you just installed.



PmacTerminal

PmacTerminal is a poor-man's command-line terminal tool like Pewin32. The panel says a lot about its operation. As the tool is explained, we will cover many tricks that you can use to build better PMACPanel applications.



Basic Terminal 101

The black "screen" is a multi-line string with a scrollbar configured for normal operation. Typed commands are sent to PMAC when you hit the <*RET*> key. LabVIEW doesn't do this automatically. To accomplish this there is a Boolean button named OK hidden behind the string control that has its Key Navigation set to capture the <*RET*> key. If you select **Project»Find** and locate "OK" you'll see it hidden there. Setting the Key Navigation this way means that when you hit <*RET*> anytime your cursor is in the window the OK button is activated. There is a lot of spaghetti diagram to keep track of the current line, character position, and other book keeping that fetches the line just entered from the multi-line string and sends it to PMAC using **PmacCommRespStr**. The terminal indicates it is expecting a command with a "->" prompt on the screen.

This process puts the Key Focus on the OK button thereby removing your cursor from the screen indicator. Focus is returned to the screen indicator by creating an attribute node for the screen indicator, selecting Key Focus, and setting it TRUE every time the OK case is executed.

Another issue that arises in a terminal-like string control is LabVIEW's use of *<TAB>* to give Key Focus to other panel controls according to the Panel Order. This can't be disabled so understand that hitting *<TAB>* throws your cursor out of the screen and onto the OK button, then the Stop button, etc. as defined by the Panel Order. Control character sequences work but don't display as you might expect. If you type *<Ctrl + A>* only the "a" appears on the screen. Hitting return does indeed send the "^a" command to PMAC and all program motion will abort as expected.

Basic Command Editing

You can use the standard cut, copy, and paste control sequences to manipulate text in the screen buffer. You can copy a previous command and paste it at the end of the buffer and execute the command with a *<RET>*. You can also copy PMAC responses to other applications or other LabVIEW string controls. You cannot insert text into the middle of previously executed commands. The spaghetti diagram doesn't know where the insert took place and would require a lot of work to track this type of operation.

If you enter lines that wrap to the next line on the screen, list programs with lines that wrap, or list the gather buffer (guaranteed to wrap) the screen may start to act a little strange. You can solve this by clearing the window using the button below the screen.

Buffer Management

PMACPanel applications, especially their development, requires the management of ASCII buffers containing command sequences, I-Variable configuration information, and motion/PLC programs. To simplify this PMACPanel has a simple VI and associated cluster control to handle ASCII text buffers. The control is located below the screen. It is a copy of the **\PmacProgram\PmacProgEdit** control shown below. The actual control on the **PmacTerminal** panel doesn't look like the raw control. We've moved things around to look better. In fact, most of the packaged PMACPanel clusters have been rearranged. Be very careful when moving items in a cluster: do NOT pull them out of the cluster boundaries by mistake. This causes the remaining controls in the cluster to reorder their Panel Order. The associated VI will not work as you expect because what was control 4 is now control 5 etc.

The cluster and VI implement six operations using the screen buffer.

- **Load** - Load an ASCII file into the screen buffer.
- **Save** - Save the contents of the screen buffer to the file specified in the path control above the screen.
- **Save As** - Save the buffer by selecting a new file.

- **Clear Window** - Dump the buffer and reset the display bookkeeping.

- **Down Load** - Save the screen to a temporary file, compile the file, and down load it to PMAC. This means that the entire buffer must contain legitimate on-line commands and/or a program. Buffers with previous PMAC responses are not downloadable.

- **Show Log** - When compiling the buffer PComm32 generates a log file with standard compiler messages. If the download generates an error, a dialog with the log file is displayed. You can use this button to review the log file whether or not an error was generated.

Terminal Indicators

There are several indicators introduced in Chapter 4 on the panel that display the status of the motor, coordinate system, and motor motions. These are simple rearrangements of the stock clusters. The interesting thing about these clusters is that they track the commands entered by you on the terminal. When you address motor 3 by typing

#3

in the terminal screen, the motor clusters display the status for motor 3. When you address coordinate system 2 by typing

&2

in the terminal screen, the coordinate system status clusters display the status for coordinate system 2. We will discuss the simple implementation of this capability a little later.

Terminal Controls

The **PmacGlobalBufferSize** VI is used to monitor and control PMAC's buffer status. Sometimes when using a terminal program its not obvious that a download can't succeed because you left a buffer open or forgot to delete the gather buffer. The size indicator and the Close Buffer button make it easier to monitor and deal with this problem.

There are three new control clusters and capabilities used to develop this tool. We will cover their use in **PmacTerminal** a little. Operationally these VIs are

- **PmacProgram\PmacProgSelect** - This menu ring control with its VI query PMAC for a list of currently loaded motion programs and display the data in the ring. Using it, you can easily determine what programs are currently loaded in PMAC.

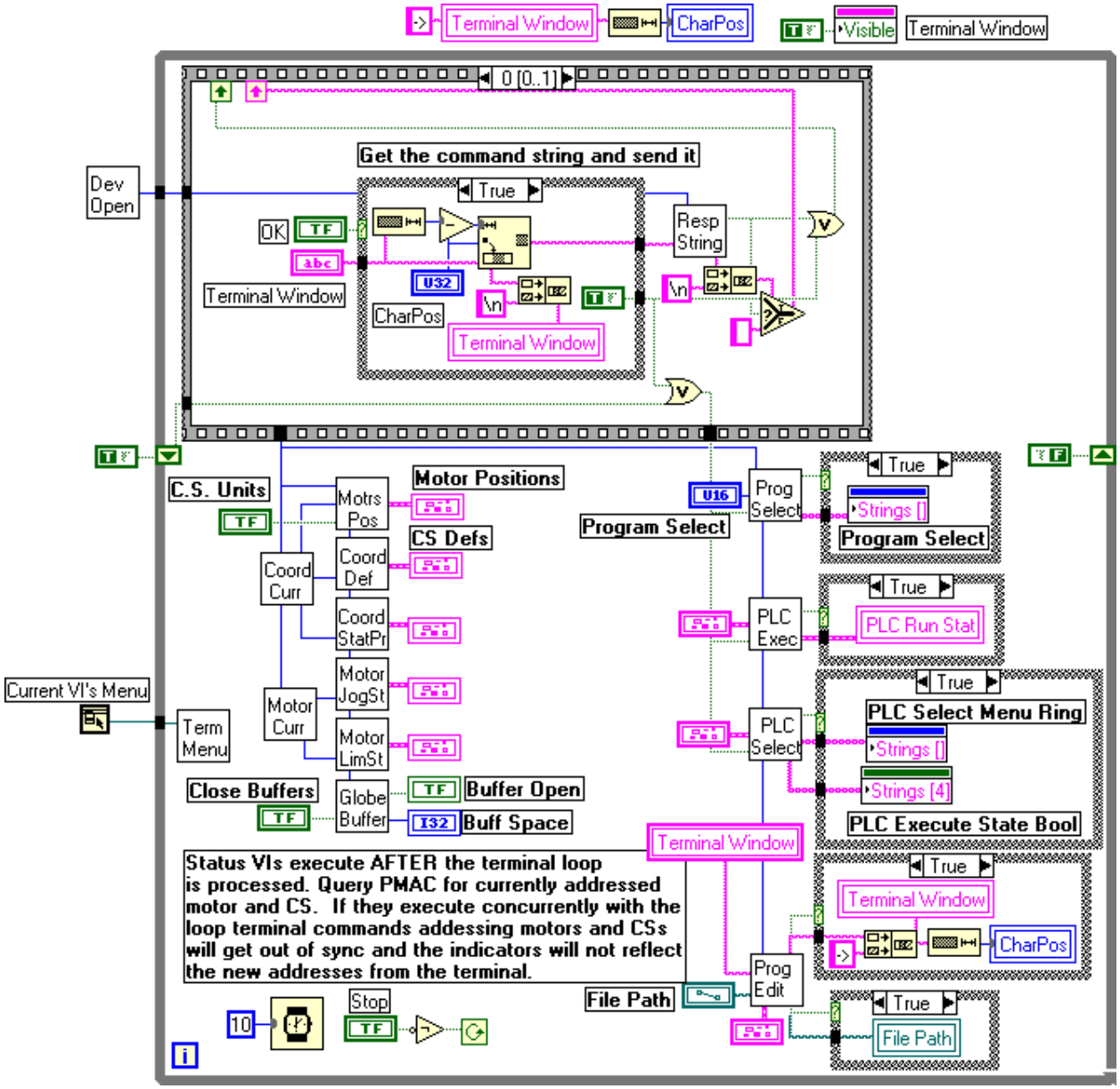
- **PmacPLC\PmacPLCSelect** - This cluster contains a menu ring control that works with its VI to query PMAC for a list of currently loaded PLC programs and their execution status: Enabled/Disabled. There is also a button that toggles the Execution State of the selected PLC. If the menu ring shows PLC 01 as Enabled the button allows you to disable it thereby changing the text displayed in the menu ring.

- **PmacPLC\PmacPLCExecute** - I5 controls the general execution of PMAC PLCs. Bit 0 controls the execution state of PLC 0 (the foreground PLC) and Bit 1 controls the execution state of PLCs 1 - 31 (the background

PLCs). This control and its VI maintain I5. When the button says Background OFF the background PLCs are off. Clicking on the button toggles them on and will now indicate Background On.

Implementation Diagram

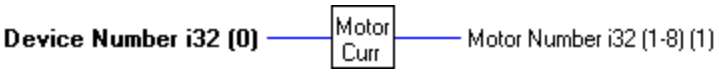
The diagram for this tool illustrates some important concepts in the development of tools and your own applications. We will not cover the spaghetti diagram in the sequence frame at the top that handles the bookkeeping for the terminal screen. The diagram is shown here



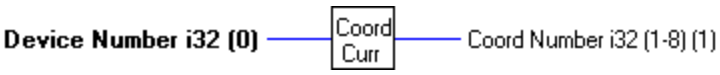
The general framework should look familiar. There is an execution loop, a **PmacDevOpen**, a Stop button, and the **PmacTerminalMenu** items. Several bookkeeping local variables are initialized outside the loop.

Most of the status indicator VIs are located in the lower left. The currently addressed motor and coordinate system are fetched by the VIs

- **PmacMotorCurrent** - Query PMAC for the currently addressed motor. It is most generally used in interactive development environments rather than a custom VI where you want to address a specific motor.



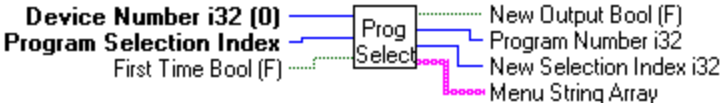
- **PmacCoordCurrent** - Query PMAC for the currently addressed coordinate system. It is most generally used in interactive development environments rather than a custom VI where you want to address a specific CS.



and provided to the six status VIs covered in Chapter 4. The Device Number they require is passed through the terminal bookkeeping sequence frame thereby causing these VIs to execute after terminal commands are processed. If this isn't done, addressing commands from the terminal get uncoordinated with the queries placed by the status VIs and the status displayed in the indicators might not be for the motor or coordinate system expected.

The VIs and spaghetti diagrams on the lower right implement the **PmacProgEdit**, **PmacProgSelect**, **PmacPLCSelect**, and **PmacPLCExecute** operations. These utilize the update architecture used in many of the earlier exercises. The VIs accept control clusters containing Booleans and generate new output data for the controls when an output Boolean indicates it has new data. Several of the clusters function as both controls and indicators using their color and Boolean text. This manual does not cover the internal operation or implementation of these VIs or members of these collections in detail. They are involved and way beyond what most developers will want to know about PMACPanel. Feel free to examine their contents and make changes as desired.

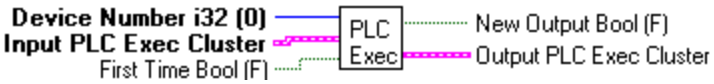
- **PmacProgSelect** - Query PMAC for a description of all loaded motion programs by reading PMAC's internal buffers. If First Time is TRUE Menu String Array contains a sorted list of all loaded programs by program number. The VI maintains New Selection Index as a state from execution to execution. Translation of Program Selection Index into Program Number occurs when First Time Strings is TRUE or Program Selection Index is not equal to New Selection Index. New Output TRUE indicates that Program Number, New Selection Index, and Menu String Array contain new data.



- **PmacPLCSelect** - Query PMAC for a description of all loaded PLC programs by reading PMAC's internal buffers. If First Time is TRUE Menu String Array contains a sorted list of all loaded PLC programs and their

execution state by PLC number for the menu ring in PLC Select Cluster. Button String Array contains information to change the description of the button in PLC Select Cluster so that it toggles the selected PLC's execution state when clicked. The VI maintains New Selection Index as a state from execution to execution. Translation of menu ring selections in PLC Select Cluster into PLC Selected Cluster occurs when First Time Strings is TRUE or either control in PLC Select Cluster changes. New Output TRUE indicates that PLC Selected Cluster, Menu String Array, and Button String Array contain new data.

- **PmacPLCExec** - This VI controls the execution of foreground and background PLC programs by modifying i5 using a PmacPLCExec control as both an indicator and a control. When First Time is TRUE New Output is TRUE and Output PLC Exec Cluster indicates the state of foreground and background PLC program execution. When either button in Input PLC Exec Cluster doesn't match the last Output PLC Exe c Cluster contents the execution state of the foreground or background PLC programs is toggled.



- **PmacProgEdit** - Manage common editing operations on Input Buffer String as specified by Program Edit Cluster. Input File Path is the default file path to use for Load, Save, or Save As operations. New Output Buffer is TRUE when a Load or Clear Window operation puts new data in Output Buffer. New Path is TRUE when a Load, Save, or Save As operation modifies the file path.

  Load - Load a file into Output Buffer.

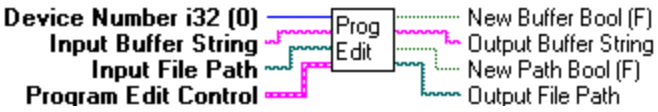
  Save - Save Input Buffer to Input File Path.

  Save As - Query the user for a new file to save Input Buffer.

  Clear Window - Put an empty string in Output Buffer.

  Down Load - Compile and down load Input Buffer to PMAC.

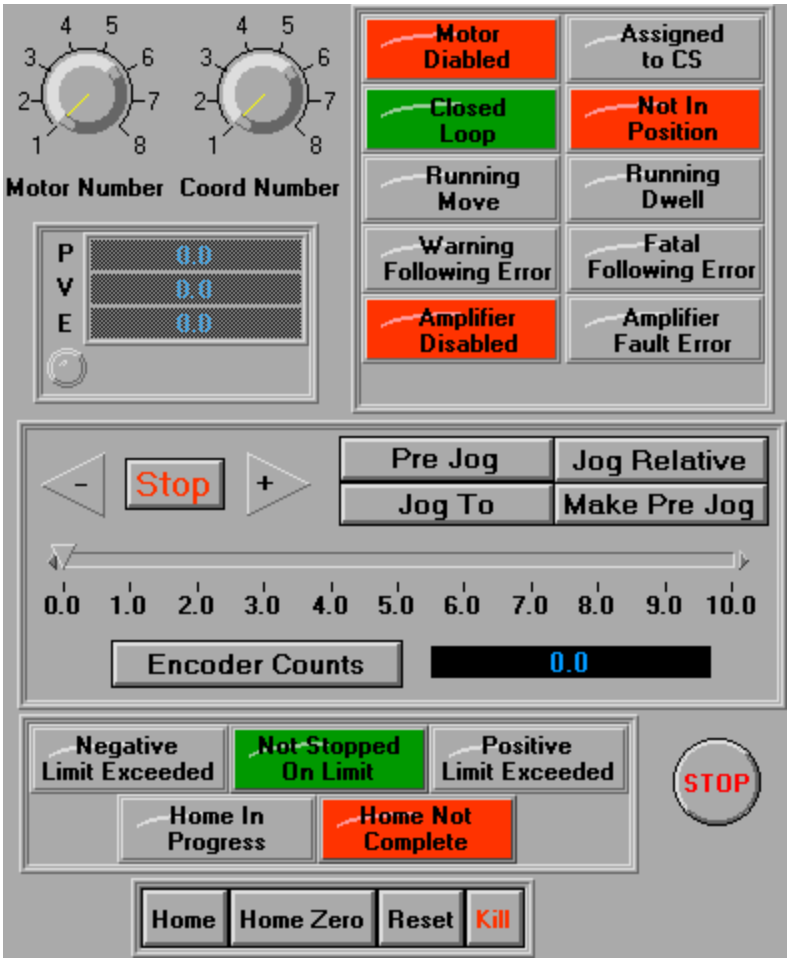
  Show Log - Display the contents of the compile log.



Using these descriptions, it is straightforward to use these VI's powerful capabilities. Place the appropriate control or controls on the panel. Where required create a writeable local variable copy of the control or the required items attribute node. If the menu ring requires an attribute node to display the programs, or the button requires an attribute node to change its Boolean text, create the node, and select the proper attribute. For the PLC Select cluster, you need to go to the panel and create the attribute node for each item in the cluster - not the cluster itself.

PmacTerminalJog

This tool is a modified version of **PmacMotorMoveExamp** and **PmacTutor7**. Its details won't be covered here. Instead, it is an excellent example for demonstrating the behavior of multiply executing tools and application VIs. One note of importance is the limits on the numeric slider in the **PmacMotorJogControl** cluster. You should change these to reflect the limits of your mechanical setup.



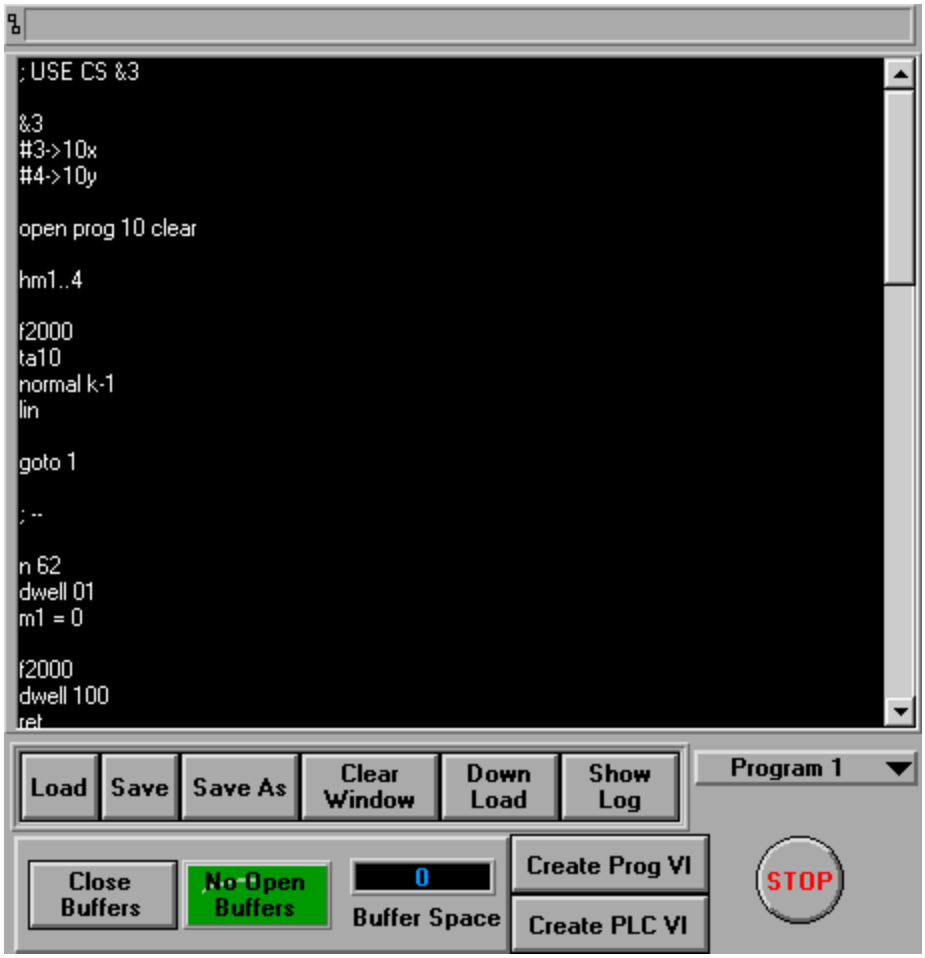
Start by opening and executing **PmacTerminal**. As you address motors and coordinate systems, the captions in the terminal tool indicators change. If you execute **PmacTerminalJog** by opening and running it or by selecting **PMACPanel»Jog** from the VI menu, things start to behave strangely. Lets say your last few **PmacTerminal** commands addressed motor #3 and coordinate system &2. Everything looks fine on the indicator bars. When you run **PmacTerminalJog,** the indicator bars suddenly reflect the status of motor #1 and coordinate system &1. Why? You never sent a terminal command to do this - did you? The truth is you did. There is only one PMAC and any commands that require a motor or coordinate system address change the addressed item. Because **PmacTerminal** queries PMAC for the currently addressed motor and coordinate system, it will use these values for its status queries. If you replace these VIs with numeric controls, this will no longer be a

problem for **PmacTerminal**. However, it will not be possible to automatically have the indicators track the commands **PmacTerminal** sends to PMAC.

PmacTerminalEdit

If you strip out all the fancy stuff from **PmacTerminal** and leave a screen, edit control, and program menu ring you get a program editor tool. The terminal like interface is different in that there is no "OK" button to capture and process the line just entered. Instead, hitting <*RET*> in the screen puts a <*CR*> in the buffer. The <*TAB*> will still rotate you through the controls.

This tool should be used to develop programs rather than **PmacTerminal**. Program development is detailed in the *PMAC User Manual* and *PMAC Software Reference Manual*. There are some added features to PMACPanel's processing of motion and PLC programs developed using this tool. PMACPanel parses the motion program buffer for the PLC or motion program number and coordinate system prior to down loading. This allows you to write a PLC or motion program and have PMACPanel keep track of this information without you having to do it. It greatly simplifies your PMACPanel diagrams. The program shown in the panel demonstrates what is required to do this.



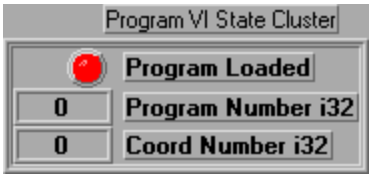
The motion program coordinate system is specified by the comment line

```
; USE CS &3
```

The down load compilation process ignores comments so this does not affect normal PMAC operation. The comment can be any mix of upper and lower case. For safety always use caps. The program number is parsed from the line

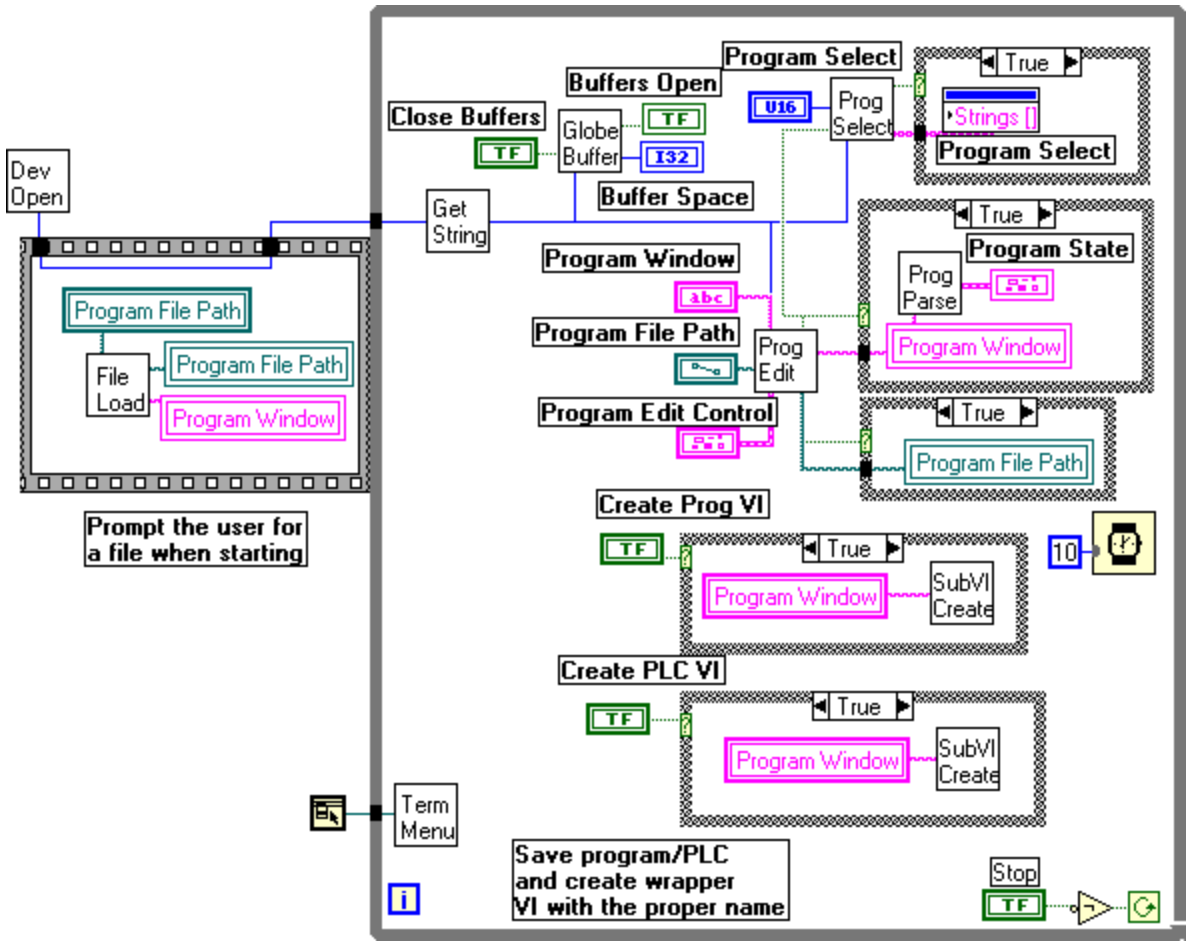
```
open prog 10
```

Again, this can be a mix of upper and lower case. When the program is down loaded a cluster indicating the coordinate system to use, program number, and Load State is created



This cluster is used by a number of **PmacProgram** VIs to do useful things for you.

The diagram for the tool has a standard organization. Note the use of **PmacProgParse** to parse the program window and generate the Program State cluster. This cluster is hidden on the panel and not used in the tool.

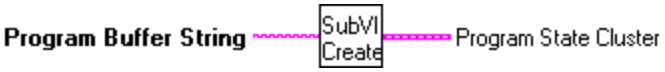


The next section discusses one of the most powerful features of PMACPanel - the encapsulation of PMAC motion and PLC programs into a single easy to use wrapper VI.

Encapsulating Motion Programs

The Create Prog VI and Create PLC VI case structures at the bottom pass the program buffer to a very special VI that creates a VI wrapper for the program being edited. The detailed use of your new motion or PLC program VIs will be covered in Chapter 7, 8, and 9.

- **PmacProgSubVICreate** - Prompt the user for a file to save Program Buffer to. Make a copy of PmacProgSubVI VI changing the name to the same base as the saved program. For example, if Program Buffer is saved to **PmacTest1.pmc** a new VI named **PmacTest1.vi** is created from **PmacProgSubVI**.



Using the example names above there are now two items with the same base name. An ASCII program file **PmacTest1.pmc** and a VI

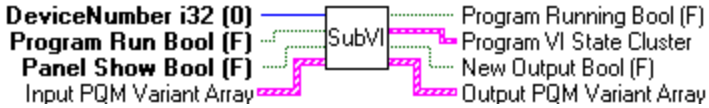
PmacTest1.vi. The icon for the new VI based on **PmacProgSubVI** shown here

- **PmacProgSubVI** - PmacProgSubVICreate makes a copy of this VI with a new name that matches the name of a motion program. Because the motion program has the same name (with a different extension) this VI knows how to open, parse, load, and run a motion program without intervention or extra inputs. It allows you to edit the associated program and interactively execute the program. Details of its implementation are contained in the manual.

  The VI downloads the associated program when first loaded unless this option is disabled in the diagram and defaults for Program Number and Coord Number are provided for the Program VI State Cluster.

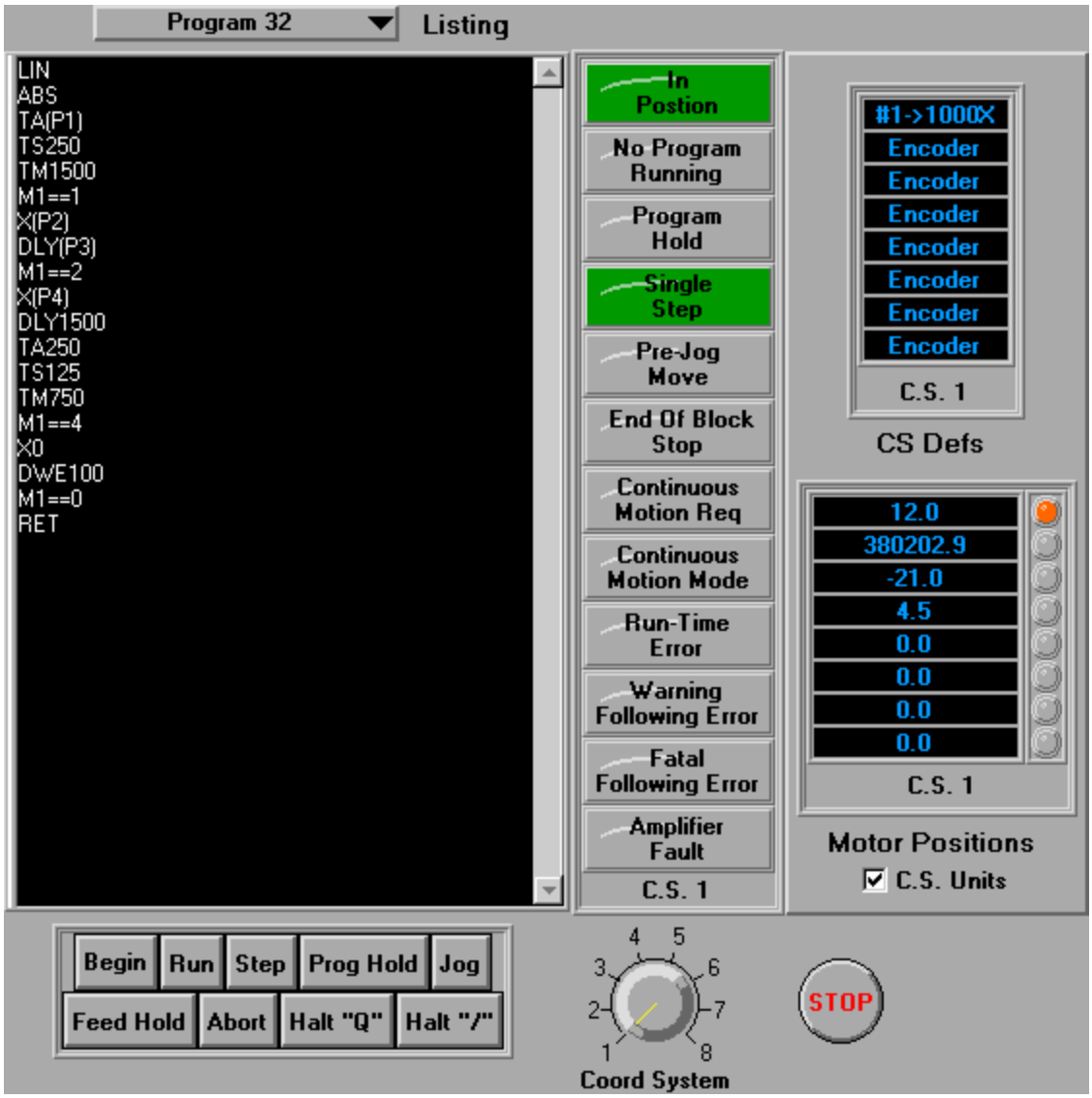
  The interactive panel can be opened and used by setting Panel Show (latched) TRUE. See the documentation for PmacTerminalEdit and PmacTerminalExecute for details on interactive execution. The panel is closed by clicking the Stop button on the panel

  When the latched input Program Run is TRUE Input PQM Variant Array is sent to PMAC to initialize a program's P, Q, or M variables. The program is then started as long as there is no program executing in the associated CS. When Program Running is TRUE this or another program is executing in the associated CS.



PmacTerminalExecute

This tool is an interactive motion program debugger that queries PMAC for the actual program and allows you to step through it in a variety of ways. Querying PMAC for the program allows you to see whether your program is actually what you think it is. If you down load a program without clearing the program buffer first the down load contents are appended to the existing buffer. This is probably not what you want. The indicators and their operation should be obvious. Extensive on-line help is available for the panel controls using LabVIEW's **Help»Show Help** facility.

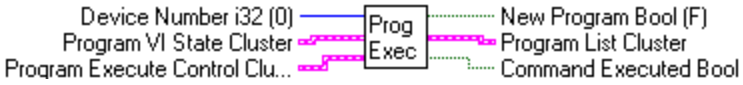


To operate the tool, select a program using the menu ring and a coordinate system to execute the program in using the knob. Only loaded programs are shown in the ring. Clicking Begin and the Program Execute cluster loads the program into the list buffer from PMAC. If you click Run, the program will begin executing. You can watch the execution by clicking the cursor in the screen. The currently executing program line will highlight and track the program through its steps. This dynamic display only works when the cursor is in the screen.

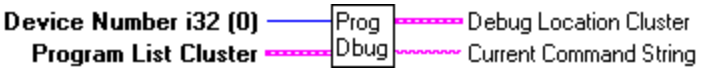
You should understand a few issues about PMAC program execution. PMAC pre-computes moves several lines ahead of the currently executing motion to allow motion blending. Because of this, the hi-lighted line may not reflect the moves your machine is currently executing. For a detailed discussion of this, see the *PMAC User Manual*.

The implementation of the tool uses two VIs to generate program execution commands and monitor the currently executing program line. These VIs are,

- **PmacProgExec** - Interactively execute the program specified in Program VI State Cluster in response to button clicks in Program Execute Control Cluster. New Program is TRUE when Begin is clicked and Program List Cluster contains a new listing. Command Executed is TRUE when any button in Program Execute Control Cluster is clicked.



- **PmacProgDebug** - Query PMAC for currently executing line in Coord Number and output the response in Current Command. Determine the scroll position and characters that delimit this line in List Buffer and create Debug Location Cluster for setting selection attributes in a multi-line string control for real-time display of Program Number's execution. This information is obtained from PMAC using the LIST PE command.



The specification of the currently executing line in the program-listing buffer is given by

Debug Location Cluster Cluster of information for string control attributes. The items define the Scroll Position of the string in the buffer, and the Start and End Character of the line currently executing.

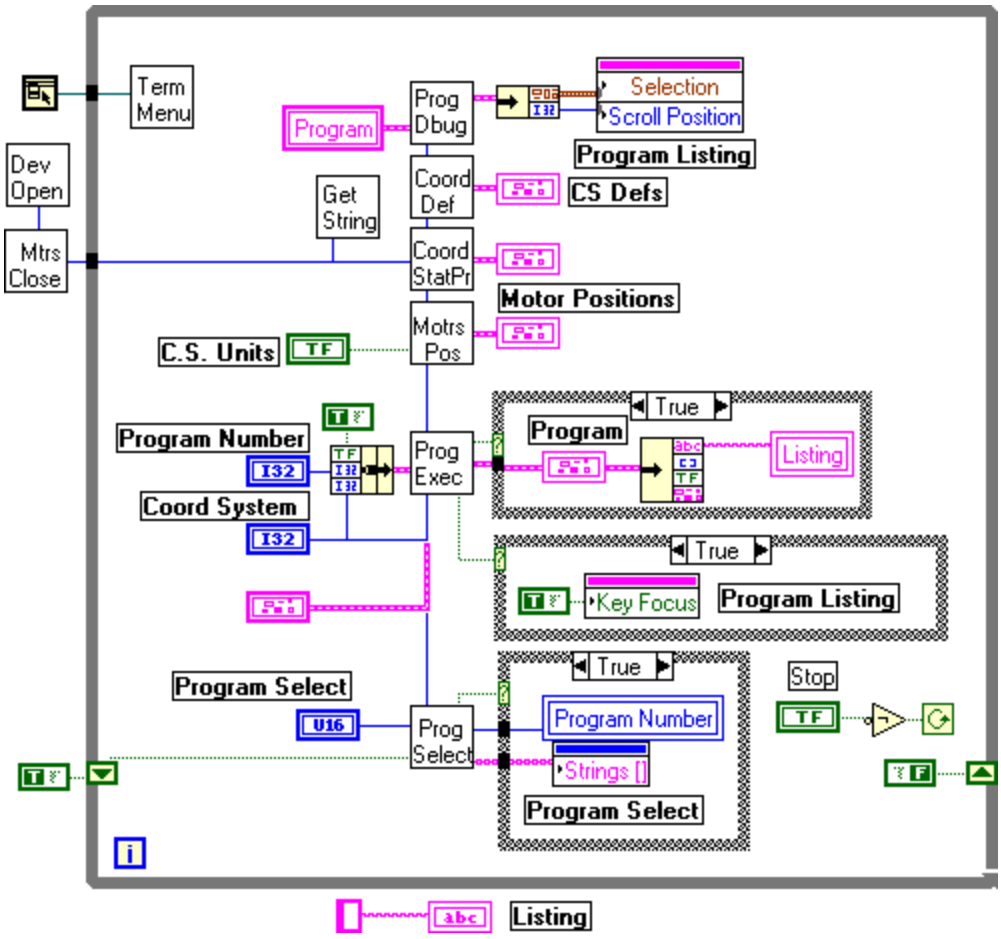
Selection Start and End character in List Buffer for currently executing program line.

Character Start

Character End

Scroll Position Number of currently executing line in List Buffer.

This information in this cluster is used in the diagram to set the selection and scroll position attributes for the string control used to display the listing. You can see this at the top of the diagram.



The program controls implemented by the Program Exec cluster send on-line program execution commands to PMAC. A brief description of the button operations is given here. For detailed descriptions of PMAC's implementation of the command see the associated documentation in the *PMAC Software Reference Manual* or the online help available through **Help»Show Help**.

- **Begin** - Point PMAC to the coordinate system and program number specified. Load the actual program from PMAC. The command sent to PMAC is **&*CS*b*PROG*** where *CS* is the specified coordinate system number and *PROG* is the current program number.

- **Run** - Execute the program from where it is. If you pointed to the beginning with Begin then start there. If you abort or halt motion using the associate buttons you can restart the program from its current location. The command sent to PMAC is **R**.

- **Step** - Execute a step to the next move or dwell in the program performing all the intervening computations. The command sent to PMAC is **R**.

- **Prog Hold** - Bring the coordinate system velocity to zero thereby holding moves where they are but allowing jogs. You can restart the program with Run or Step. The command sent to PMAC is \.

- **Jog** - Pop a jog panel up to allow motor jogging after a Prog Hold. The jog panel must be closed before the panel before you can return to this panel to restart the program with Run or Step. Closing the panel returns all jogged motors to their pre-jog position with **j=**. Commands sent to PMAC are generated from the panel.

- **Feed Hold** - Bring the coordinate system velocity to zero thereby holding moves where they are. You cannot jog from this mode. You can restart the program with Run or Step. The command sent to PMAC is H.

- **Abort** - Decelerate all motors immediately. You can restart the program with Run or Step. The command sent to PMAC is **A**. See the safety notes for the command in the *PMAC Software Reference Manual*.

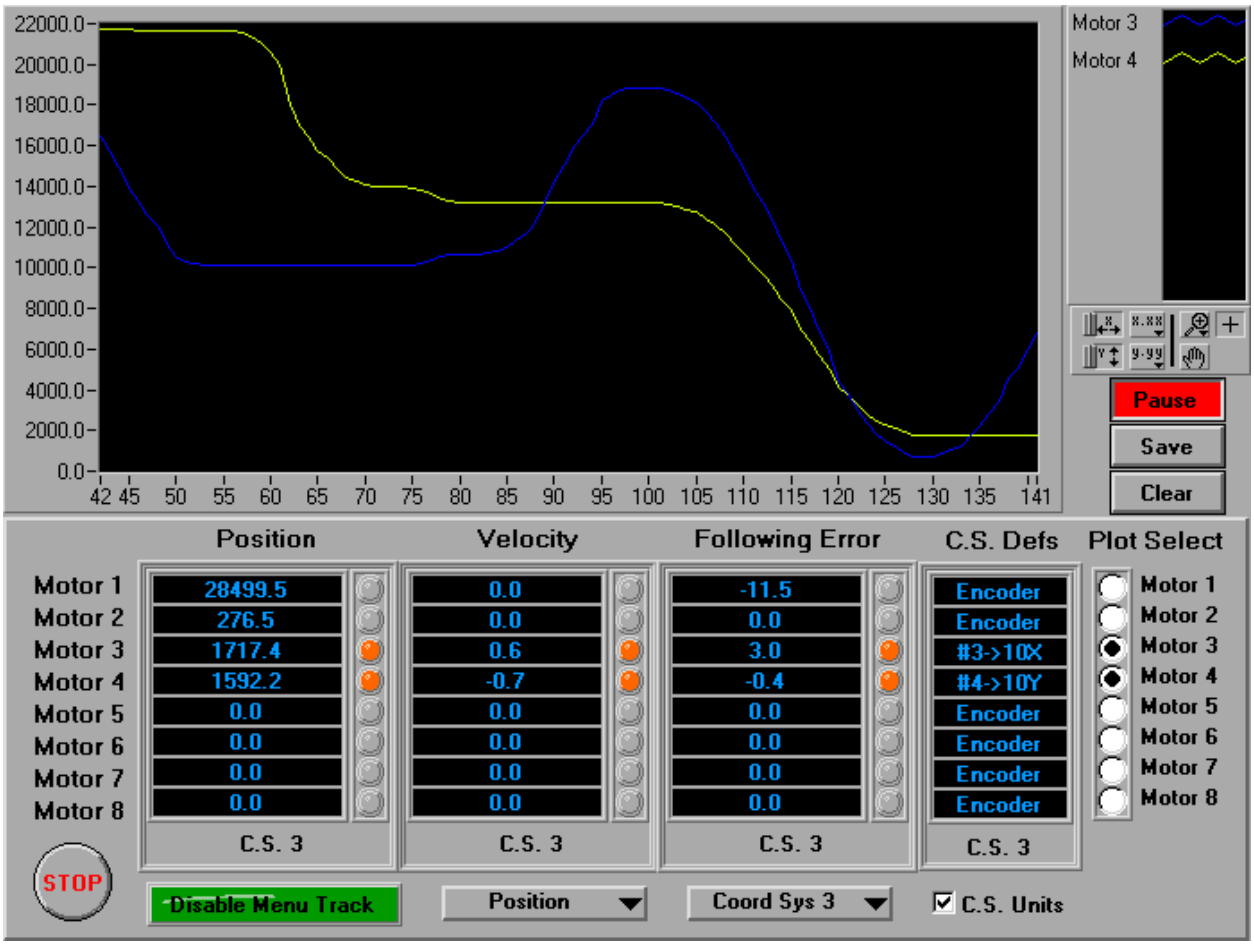
- **Halt "Q"** - Quit the program at the end of the current move or any already calculated moves. You can restart the program with Run or Step. The command sent to PMAC is **Q**.

- **Halt "/"** - Quit the program at the end of the currently executing move. You can restart the program with Run or Step. The command sent to PMAC is /.

PmacTerminalMotors

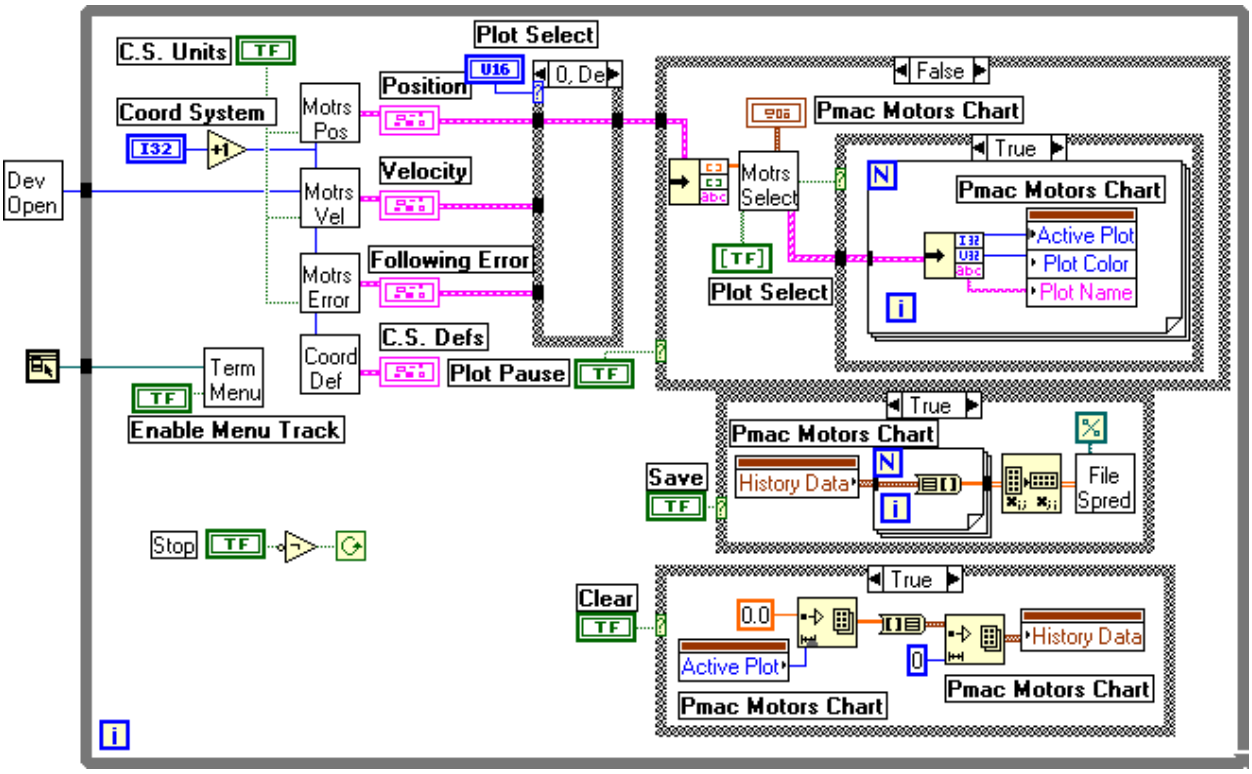
This tool is based on **PmacTutor10** that introduced the **PmacMotors** ICVs. Its purpose is to allow you to monitor selected motor motions from the terminal. The arrays of indicators function as introduced in the earlier tutorial. You can select a coordinate system and enable C.S. conversion for motors defined in the specified coordinate system. Motors selected in Plot Select are displayed in the real-time chart. The sampling rate is not uniform. It is a best-effort attempt to query as fast as possible. For uniform sampling use **PmacTerminalGather**.

You can choose to display motor positions, velocities, or following errors. The chart control palette allows you to pan and zoom the display. The Free Run/Pause button allows you to pause the chart update. Clear will clear the chart and restart the chart history buffer. The Save button will prompt you for a file to save the chart data as a tab-delimited spreadsheet file. You can also select **File»Print Window** from the menu bar to print the panel to the printer or a file at any time. Servicing **PmacTerminalMenu** may cause noticeable glitches in the plot data. It can be disabled by clicking the Disable Menu Track button.



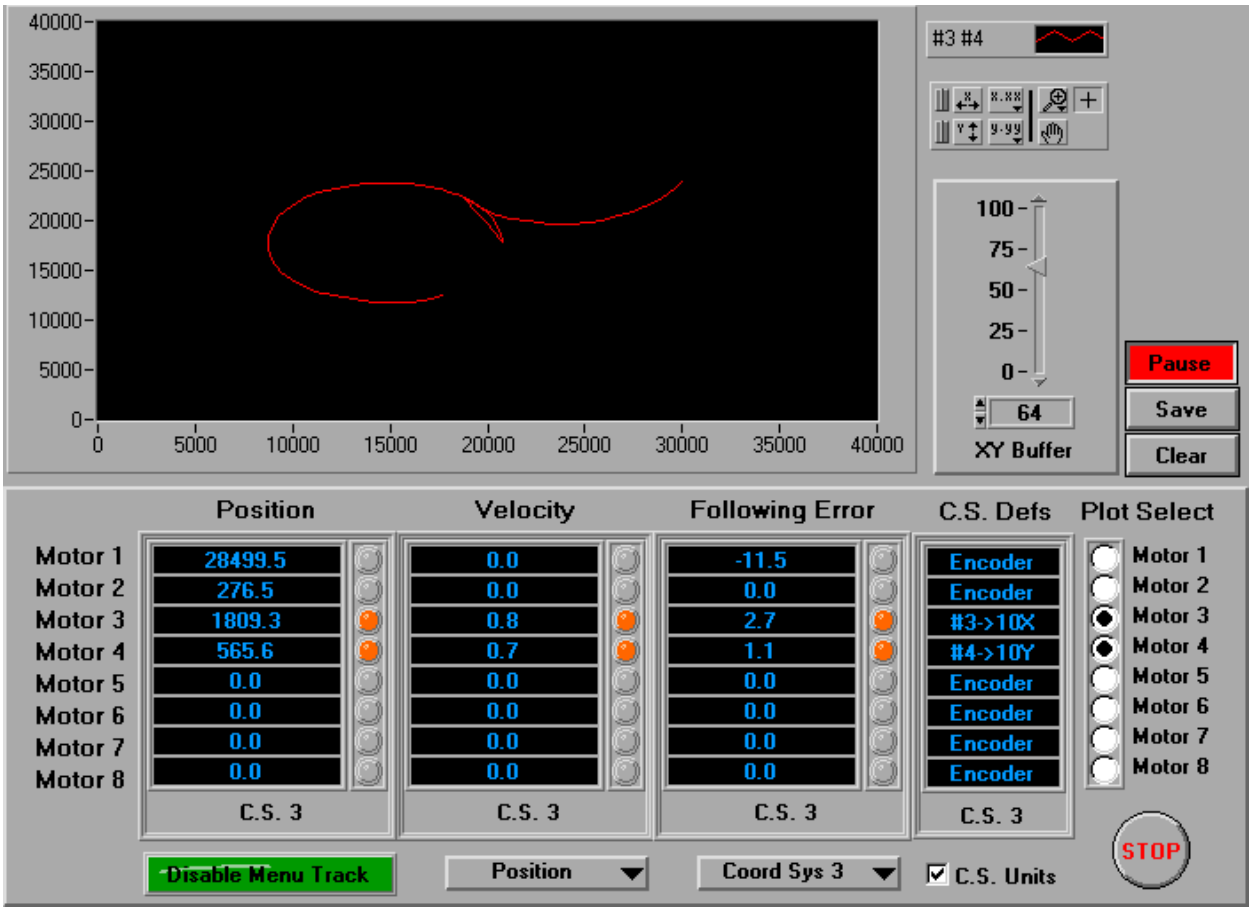
Depending on your requirements, you should alter chart operation using the attributes available with the right mouse button. These include the length of the history axis, auto scaling of the Y-axis (P, V, or E), plot style, and many other items. If you want to change the range on the Y-Axis click the minimum or maximum value on the axis, enter a new value, and disable the auto scaling. Don't forget to save these changes if you want them to be permanent.

The diagram has a case structure to allow selection of positions, velocities, and following errors. If you wish to plot a mix of these, you should modify the diagram. Pause is implemented with a case structure that prevents the update of the chart. If you desire, the VIs for querying position, etc. and the PmacMotorsPVE clusters can be moved into the pause case. This way the indicator update will also pause. Save and Clear use the chart attribute nodes to access the History data required to implement their operations. You will find these pieces of the diagram useful in many other applications because their use is not obvious in the LabVIEW documentation.



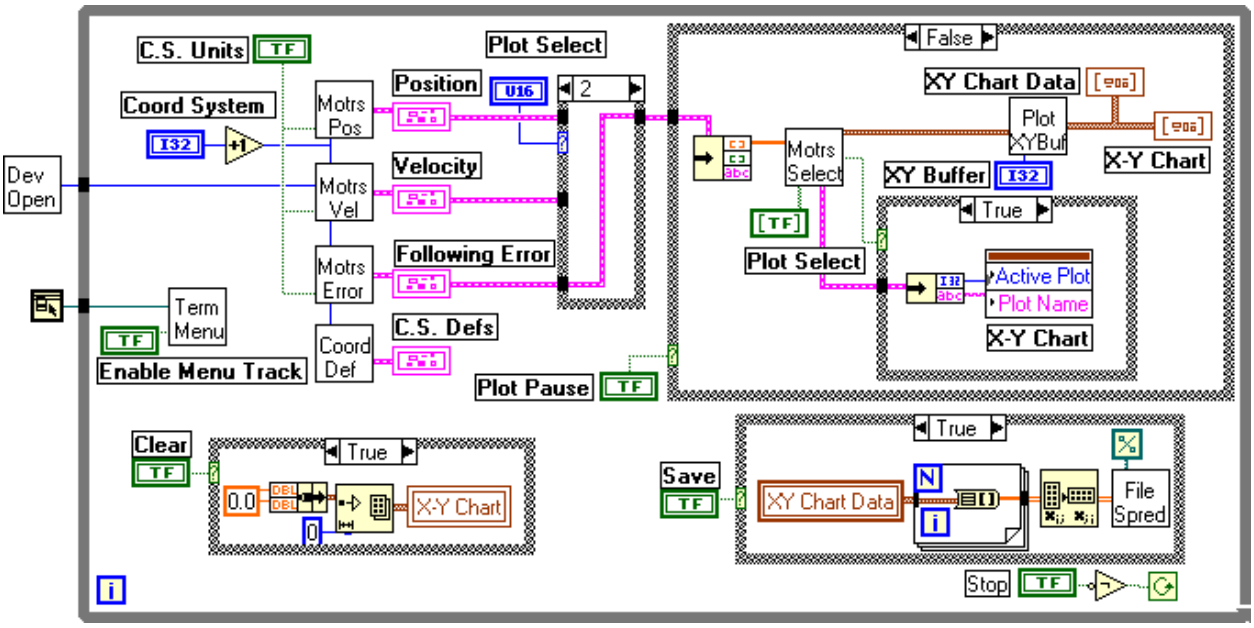
PmacTerminalMotorX-Y

This tool is based on **PmacTerminalMotors**. Its purpose is to allow you to monitor and plot selected motor motions executing in a 2-Axis Cartesian coordinate system. Ninety-five percent of this tool functions as introduced in **PmacTerminalMotors**. See the documentation for this tool. The difference is that motors selected in Plot Select are displayed in the pseudo real-time X-Y chart. To display 2-Axis data you must select two motors.



The charting of 2-Axis motion uses a special chart buffer **PmacMotorsPlotXYChartBuffer** to fool the standard LabVIEW X-Y plot control into behaving like a real-time chart. This is required because LabVIEW's X-Y plot does not maintain a history buffer - it plots sets of points. The slider on the right of the plot specifies the length of the history buffer for **PmacMotorsPlotXYChartBuffer**. Like a standard real-time chart, 2-Axis position history is displayed like a snake moving around the coordinate space. The head is current position and the tail is the last position in the buffer. This tool will definitely aid you in understanding what moves you are executing in a 2-Axis system.

The implementation of the X-Y charting is very similar to that used to implement the standard chart in **PmacTerminalMotors** and **PmacTutor10**. The main difference is that the chart has no history buffer. Hence, we have created a hidden control for the data provided to the chart named XY Chart Data.



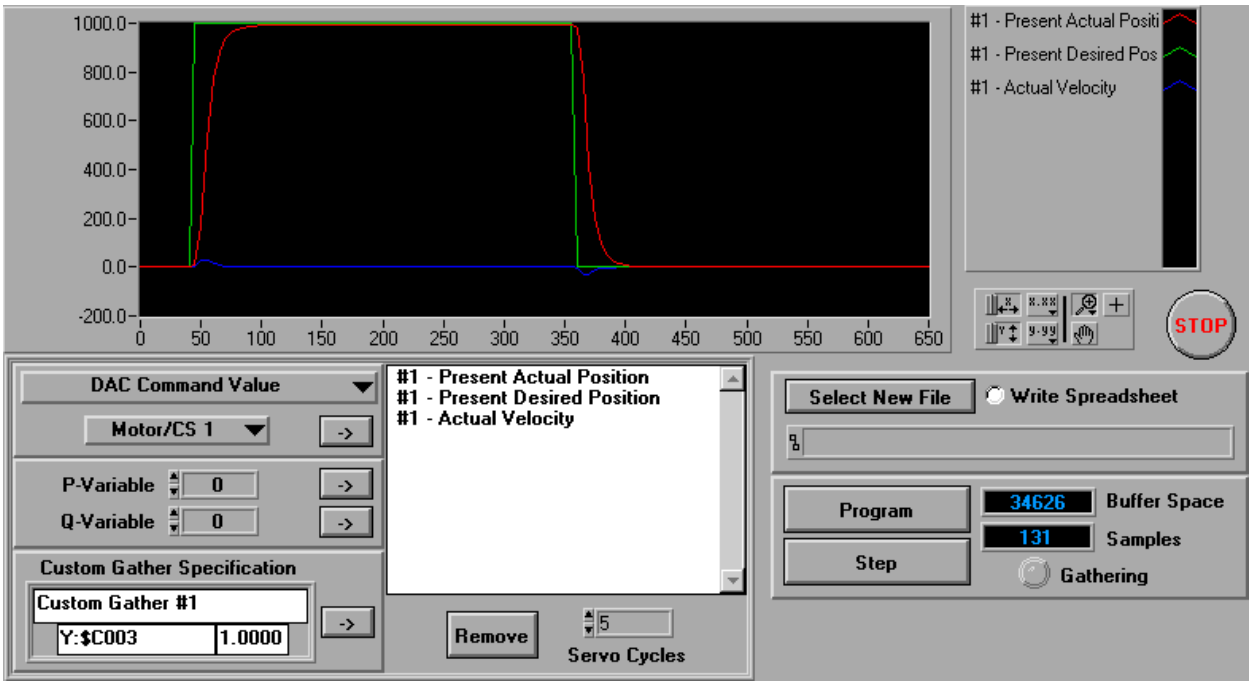
PmacPlotXYChartBuffer takes the two point X-Y cluster terminal from **PmacMotorsPlotSelect** VI and buffers them to generate an array of clusters for the X-Y Chart. To update the plot legend create an attribute node for the chart, select the two items shown, and set them from the cluster provided by **PmacMotorsPlotSelect**.

PmacTerminalGather

This tool is a general purpose tool for synchronously gathering and plotting PMAC data and outputting the data to TAB delimited spreadsheet files for use by Microsoft Excel, LabVIEW, Matlab, or other analysis and plotting applications. It allows you to execute a step or an encapsulated motion program. Encapsulated motion programs created by **PmacTerminalEdit** can easily be installed in the tool by replacing the existing **PmacProgSubVI** with one of your own.

The LabVIEW chart controls enable you to pan and zoom the plot. You can select **File»Print Window** from the menu bar to print the panel to the printer or a file at any time.

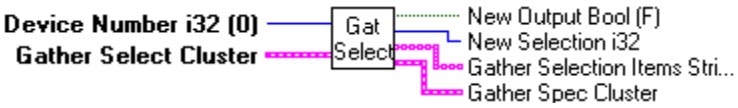
As with the **PmacTerminalMotors** and **PmacTerminalMotorsXY** tools you can modify the plot attributes prior to running the tool to accommodate your expected scale. PMACPanel has utilities to set the legend and plot colors to reflect your gather selections. One issue you should consider when gathering data is that the gathered variables can and will, as in the sample panel, have widely varying ranges. You can choose to stack the plots if you desire.



To select Address Items use the controls contained in the **PmacGatherSelect** cluster on the bottom left. PMAC lets you select up to 24 Address Items. The following section covers methods for extending the **PmacAddress** selection VIs. The **PmacGatherSelect** VI builds an array of Address Items containing a string description, an address, and a scale factor using the interactive commands from the panel cluster.

- **PmacGatherSelect** - Maintains a PmacGatherSelect Cluster and builds a PmacGatherSpec cluster to define gather operations. You can build a list in four ways. Select an item and Motor/CS number, P-Variable, Q-Variable, or define a Custom Gather Specification. Click the associated -> button to add the item to the list on the right. The gather sample rate is defined as a number of Servo Cycles. All items are gathered at the same sample rate. Items selected in the list can be deleted using the Remove button.

  New Output is TRUE when an item is added to the list with a -> button or removed from the list with Remove. New selection identifies the selected item in the gather list. Gather Selection Items String Array define the contents of the gather list. Gather Spec Cluster is an internal data type used by other PmacGather VIs to setup PMAC and collect the gathered data.



There are four methods for specifying Address Items provided by this VI and its control/indicator cluster.

- The top group of two Menu rings allows you to select one of 29 standard motor or CS variables and a motor or CS. Using the -> button in this group you can add the selection to the Text ring on the right.

- If you specify a P-Variable or Q-Variable number and click the appropriate -> button the specified variable is added to the selection list.

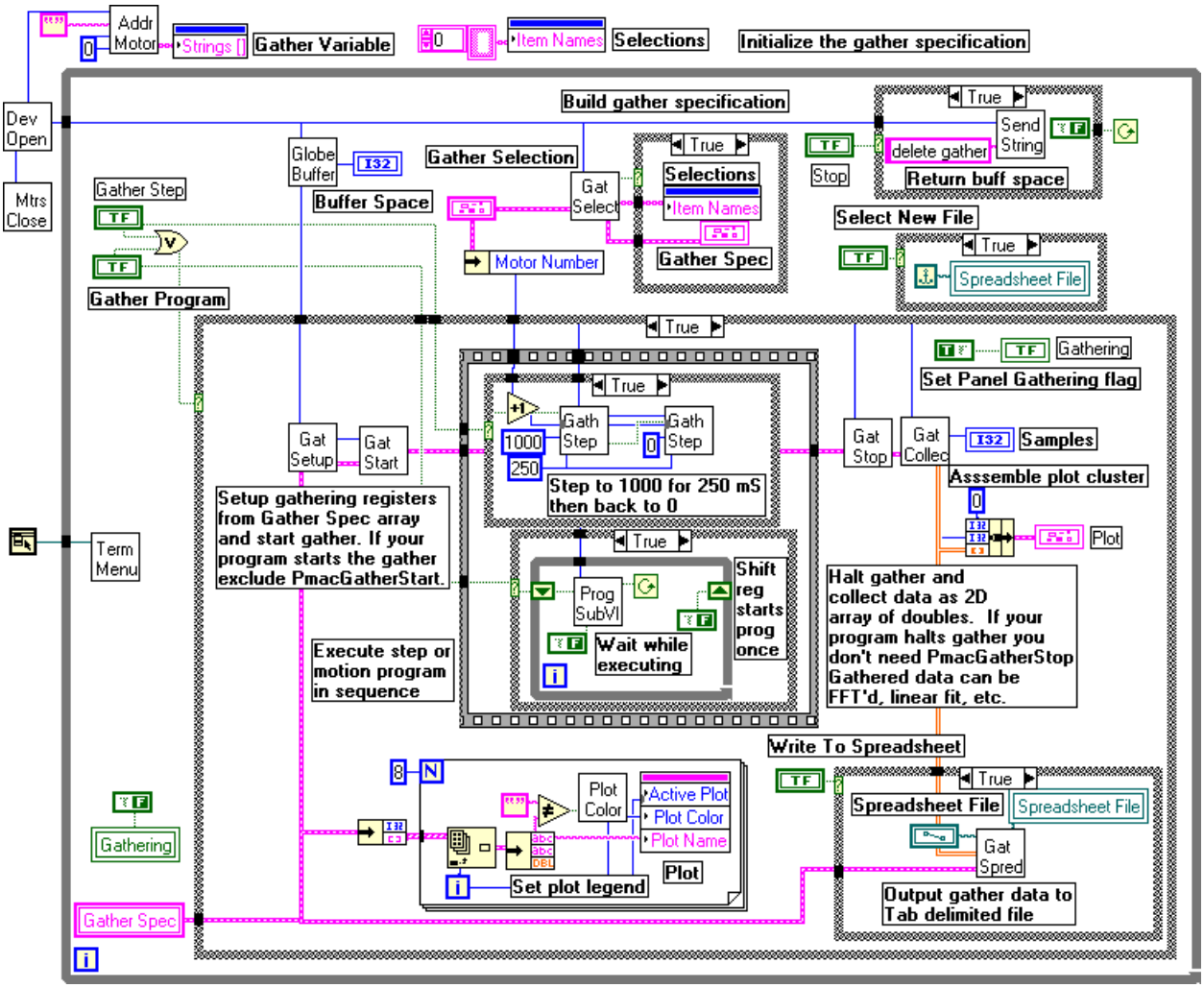
- On the bottom is a Custom Gather Specification cluster that allows you to enter a description, address, and scale factor. Clicking its associated -> button adds this item to the selection list. Be aware that the PComm32 library requires you to specify addresses as:

```
X:$****
Y:$****
DP:$****
```

If you want to write the gathered data to a spreadsheet file click the Write Spreadsheet radio button before executing the gather. To actually gather the data click Step or Program. Step executes a 250 mS step to 1000 counts and back to 0 counts using the motor specified in the menu ring used to setup the gather. Program executes the encapsulated motion program wired into the diagram. During the gather, the Gathering indicator is green. After the gather, the data is plotted and the indicator turns gray. If you selected Write Spreadsheet, you will be prompted for a file name.

The diagram for implementing this tool is a great source of ideas for building gathering into your own applications. Being a full application tool it requires a bit more work – but it is not as bad as it looks – it fits on one page!

Outside the execution loop, all motor servo loops are closed by **PmacMotorsCloseLoop**. To exit the loop the Stop button first sends a delete gather command to PMAC to free the buffer space. Buffer space is monitored using **PmacGlobalBuffers** and an indicator. We will discuss how a Gather Selection list is built in the next section.



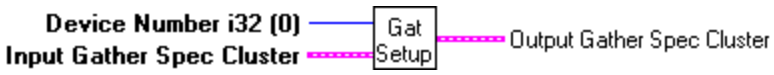
Data is actually gathered in this tool when either the Gather Program or the Gather Step buttons are clicked. They require the Gather Spec Cluster created by your program or generated by **PmacGatherSelect**. The gather process begins by executing in sequence the following five VIs. In the diagram, the sequence of operations is located in a case structure.

- **PmacGatherSetup** - Use the information in Input Gather Spec Cluster to setup a gather operation on PMAC. Output Gather Spec Cluster should be wired to PmacGatherStart, PmacGatherStop, and PmacGatherCollect to sequence operations and so that they can get the information they require for their operation.

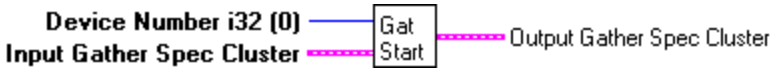
  Use the information in Input Gather Spec Cluster to setup a gather operation on PMAC. Output Gather Spec Cluster should be wired to PmacGatherStart, PmacGatherStop, and PmacGatherCollect to sequence operations and so that they can get the information they require for their operation.

  The actual setup can also be done using Pewin32, PmacTerminal, or your motion/PLC programs. This is not recommended if you intend to use PmacGatherCollect to retrieve the gathered data. These methods require

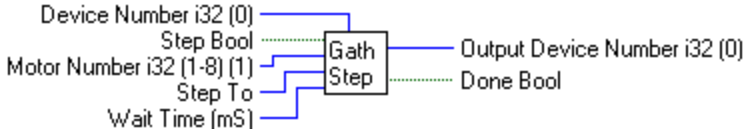
intimate knowledge of PMAC's internal architecture and are automatically handled by this VI.



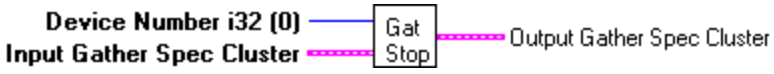
- **PmacGatherStart** - Start a previously defined gather operation. You should immediately start the desired motion after this VI executes. You can eliminate this VI if you start gathering by using the PMAC program command "define gather" in your program.



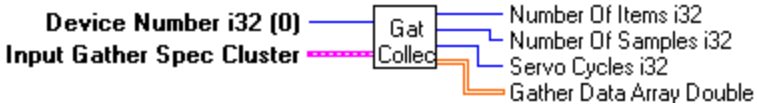
- **PmacGatherStep** - This is only one of any possible motion or encapsulated motion program.



- **PmacGatherStop** - Stop an executing gather operation. You can eliminate this VI if you stop gathering by using the PMAC program command "end gather" in your program.



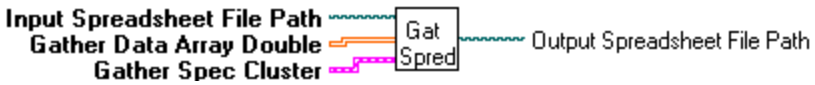
- **PmacGatherCollect** - Collect the gather buffer and scale the data using each items scale factor. The data in Gather Data Array is a two dimensional array of doubles with Number of Items columns and Number of Samples rows. In this format the data can be written to a spreadsheet or processed by many different LabVIEW data analysis VIs.



The sequence frame in the middle of the case executes the step or an encapsulated motion program. You can replace the motion program with your own or modify the entire sequence to suit your needs.

There are two other operations performed within the main case structure. The Gather Spec cluster is unbundled and used with **PmacPlotColor** to setup the legend. Finally, after the data is collected it can be written to a spreadsheet if the operation was enabled prior to the gather. This is done using

- **PmacGatherSpreadsheet** - Output a tab delimited spread sheet file for import into other plotting and analysis applications. If Input Spreadsheet File Path is empty of Not A Path a dialog prompts for a file name. The file name used is provided to Output Spreadsheet File Path.



Specifying Gather Addresses

With PMAC you can gather data from any address. This requires an address to gather from and a scale factor to apply to the data. The **PmacGather** tools use a small collection of **PmacAddress** VIs to simplify the specification of PMAC addresses for gathering. When you understand these tools, you can modify them to suit your particular needs.

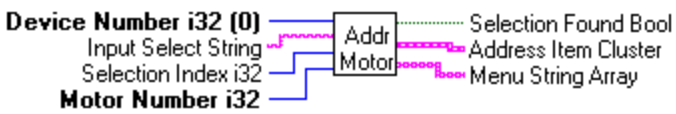
The purpose of the **PmacAddress** collection is to build arrays of Address Item Clusters as shown here to define an Address Item's text description, address, scale factor, and type

Address Item Cluster Specify a description, address, and scale factor for a Address Item

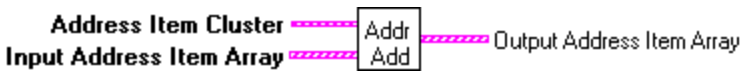
- **Address Item Description** Text description of Address Item
- **Address Item Address** Address of Address Item
- **Address Item Scale** Scale factor for Address Item
- **Address Item Type** Enumerated type defining type of raw data

The **PmacAddress** collection consists three VIs

- **PmacAddressMotors** - This VI maintains a table defining 29 of the most common Address Items. If Input Select String is the empty string the VI produces Menu String Array describing the defined Address Items. This should be used to set the items in a Menu or Text ring control. Selection Index and Motor Number are provided by rings and define the desired item and the motor number used to compute an address for the specified item. The computed item is contained in Address Item Cluster. For a description of this computation see the reference section and the memory map contained in the *PMAC Software Reference Manual*.



- **PmacAddressAdd** - Check to see if the item specified by Address Item Cluster already exists in Input Address Item Array. If it already exists do not add it. If it does not exist add the cluster to Output Address Item Array.



- **PmacAddressDelete** Locate and remove the Address Item Cluster specified by Selection Index to Delete from the Input Address Item Array.

The **PmacAddressMotors** panel contains an array of clusters that define a translation table. You can manually add items to this table and set them as the defaults for the control transparently adding them to the menu ring in the **PmacGatherSelect** control. A portion of the table provided with PMACPanel is shown here.

Translation Table Private

16

Actual Velocity	X:$	0033	3C	0.03
Integrated Error Residual	X:$	0036	3C	1.00
Integrated Error	DP:$	0037	3C	1.00
Previous Phase Position	Y:$	003F	3C	1.00
Slip Frequency	X:$	0040	3C	1.00
Present Phase Position	DP:$	0041	3C	1.00
Phase Advance	X:$	0042	3C	1.00
Phased DAC Amplitude	Y:$	0043	3C	1.00
DAC Command Value	X:$	0045	3C	1.00
Compenstation Correction	DP:$	0046	3C	0.03
Encoder Status Bits	X:$	C000	4	1.00
Encoder Time Between Counts	Y:$	C000	4	1.00
Encoder Time Since Last Count	Y:$	C001	4	1.00
Encoder Phase Position	X:$	C001	4	1.00
			0	0
			0	0

Each cluster item in the array consists of five items. In order from left to right these are:

Name Textual description of the item to be gathered. Used in conjunction with a motor or CS number to build a unique description for plot legends and spreadsheet files.

Address A string defining the size and interpretation of the data to be gathered. Legitimate designators are X:$, Y:$, and DP:$.

Address Offset A hexadecimal string defining the offset address of the data to be gathered.

Address Stride A hexadecimal numerical value that defines a stride to be used in computing the final gather address. The actual address is computed as (Motor Number - 1) * Address Stride + Address Offset.

Scale Factor A scale factor to apply to the collected data. Some entries in this table compute this value depending on the item being gathered.

You should refer to the PMAC I/O and Memory Map in the *PMAC Software Reference Manual* prior to modifying this table. You will note in the table that Encoder Time Between Counts has an Address Stride of $4 and an Address Offset of $C000, whereas most Address Strides are $3C. Using the values the address for Motor 3 Encoder Time Between Counts is computed to be:

```
Y:$(C000 + (Motor Number - 1) * 4) = Y:$C008
```

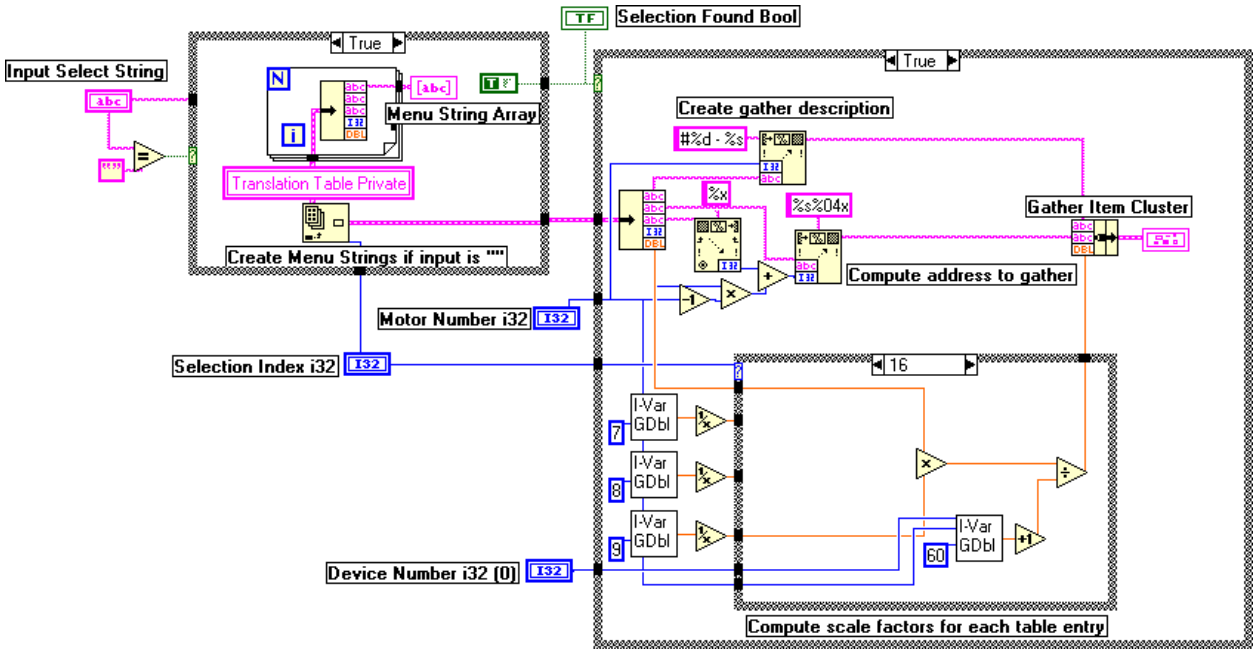
As another example, the DAC Commanded Output for motor 2 is

```
X:$(0045 + (Motor Number - 1) * 3C = X:$0072
```

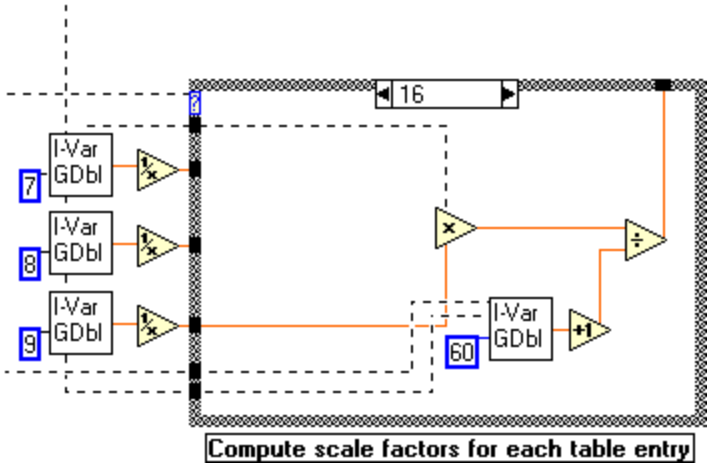
Scale Factors are a little more complex. Many Address Items in memory are scaled by one and already have the proper scaling. The most interesting ones are scaled by some combination of factors and I-Variables from internal units to encoder or coordinate system units. For example, Present Actual Position (DP:$002B) is reported in units of

```
1 / (Ix08 * 32) counts
```

To make your life easier **PmacAddressMotors** computes this scale factor when building the Address Item Cluster. If you add items to the table remember to make them the default for the table and save the VI. When you add the item, you must add a little wiring to the diagram for **PmacAddressMotors** shown here.



At the bottom of the diagram is a labeled case structure labeled "Compute Scale Factors for each table entry". It is reproduced here for clarity.



Select the last case in the structure using your mouse and add another case. The default last case is 29 so add case 30 or whatever you require. If the scale factor is fixed and specified in the table, wire the orange scale factor input tunnel from the unbundled cluster to the output select tunnel on the right. Click through a few of the cases and you'll see what we mean. This will copy the scale factor in the translation table into the scale factor item for the Address Item Cluster being built. The case shown here depicts the scale factor computation for item 16 - "Actual Velocity" at Address Offset X:$0033 with stride $3C. The units of the gathered data, as documented in the *PMAC Software Reference Manual* Chapter 8, are:

```
1 / (Ix09 * 32) counts / (Ix60 + 1) servo interrupts
```

The scale factor in the translation table is 1/32. Hence, this portion of the diagram computes

```
ScaleFactor * (1 / Ix09) / (Ix60 + 1)
```

When modifying the tables make sure that you keep an original copy of the **PmacAddressMotors** VI. If you happen to pull a control out of the cluster or pull the cluster out of the array the table clears all entries.

Chapter 6 - Encapsulated Motion Programs and PQMs

Basics

This chapter introduces a variety of VIs and tools to seamlessly integrate PMAC motion programs into your PMACPanel application. In Chapter 5, we introduced **PmacTerminalEdit**. This tool allows you to develop new motion or PLC programs or modify existing programs and with the click of a button create a VI wrapper for the program. In this chapter we cover the details of the wrapper and introduce the **PmacPQM** collection of VIs. These provide an interface to directly tie controls and indicators in your application panels to motion program variables.

PmacProgSubVI

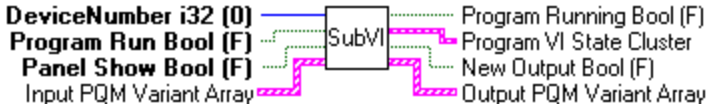
We introduced this VI in Chapter 5 but do so again because we're going to cover it in more detail.

- **PmacProgSubVI** - PmacProgSubVICreate makes a copy of this VI with a new name that matches the name of a motion program. Because the motion program has the same name (with a different extension) this VI knows how to open, parse, load, and run a motion program without intervention or extra inputs. It allows you to edit the associated program and interactively execute the program. Details of its implementation are contained in the manual.

  The VI downloads the associated program when first loaded unless this option is disabled in the diagram and defaults for Program Number and Coord Number are provided for the Program VI State Cluster.

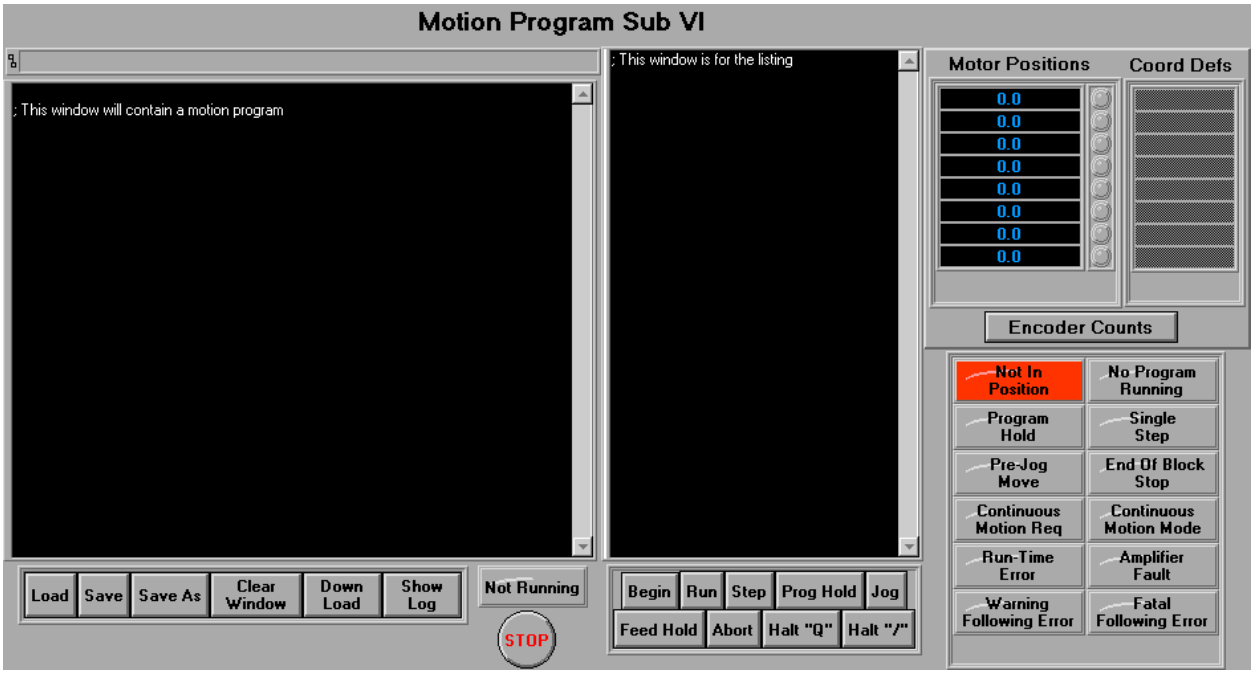
  The interactive panel can be opened and used by setting Panel Show (latched) TRUE. See the documentation for PmacTerminalEdit and PmacTerminalExecute for details on interactive execution. The panel is closed by clicking the Stop button on the panel

  When the latched input Program Run is TRUE Input PQM Variant Array is sent to PMAC to initialize a program's P, Q, or M variables. The program is then started as long as there is no program executing in the associated CS. When Program Running is TRUE this or another program is executing in the associated CS.



When **PmacTerminalEdit** saves your motion program to a file and makes a copy of **PmacProgSubVI** with the same name as your motion program you have encapsulated the program within a VI. You should edit the icon of your new encapsulation or wrapper VI to represent your motion program. We will use the terms encapsulation and wrapper interchangeably. Before we look at how to use the encapsulation VI lets look at the new VIs panel and diagram.

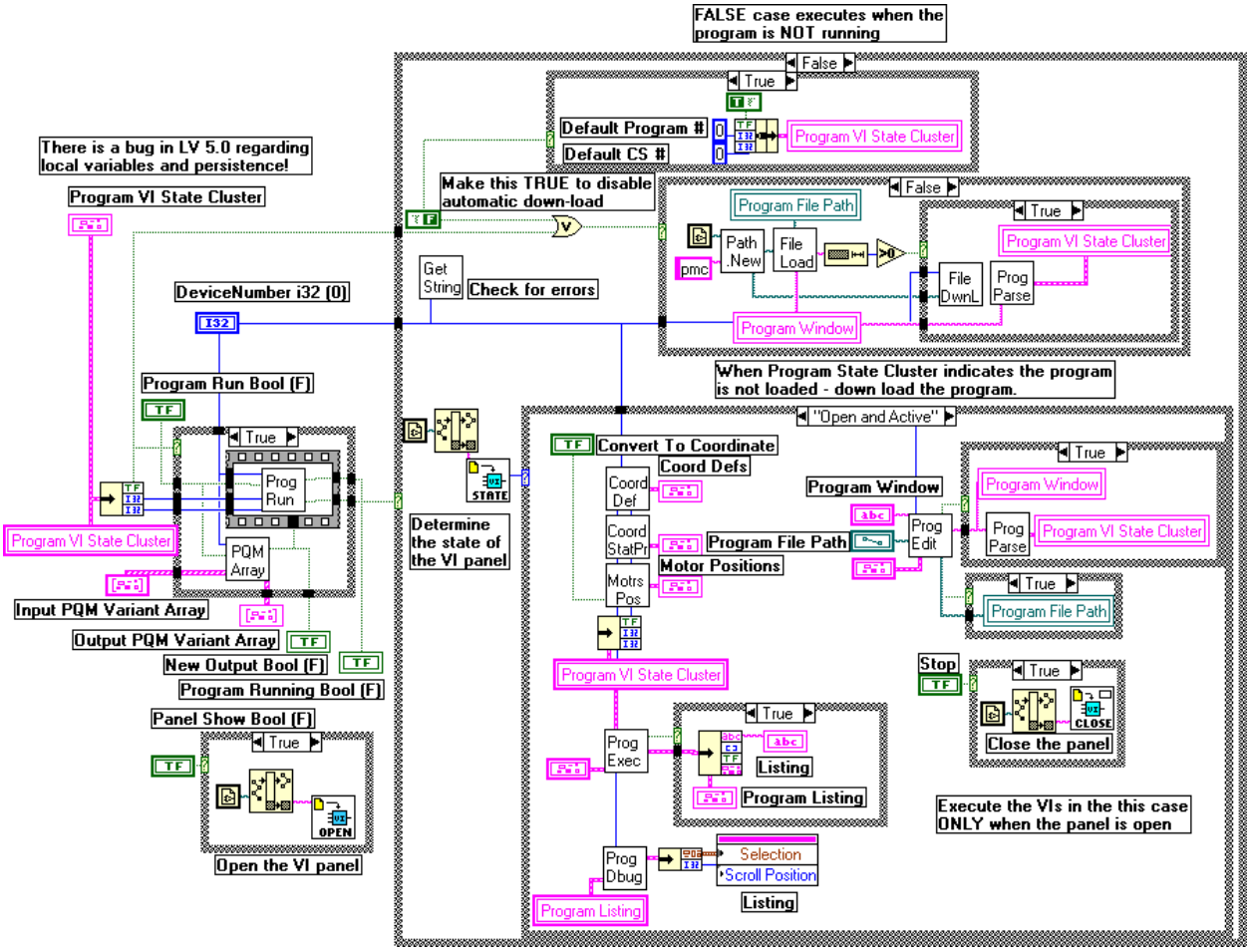
If you open your new encapsulation VI you'll note that the panel is a combination of **PmacTerminalEdit** and **PmacTerminalExecute** with most of their capabilities. The purpose of this panel is to allow you to edit the associated motion program and monitor its execution. There is no need for buffer control or for selecting a motion program or coordinate system because these are already known by your encapsulation VI. Refer to their documentation in Chapter 5 for details on using the capabilities of the two panel components. **Help»Show Help** will also provide detailed descriptions of the buttons and indicators.



The implementation of the VI is quite a bit different from most of those already introduced. This VI is embedded in your application's execution loop so that it can continuously monitor the attached motion program. As such, it is not wrapped in its own loop. It utilizes several VI control and server concepts found in LabVIEW to control the display of the panel and selective execution of some of its components so that it doesn't consume a lot of execution time unless required.

There are four major pieces of the diagram. On the far left is a case structure that controls and monitors the actual execution of the attached motion program. Below this is a small case structure that opens and displays the panel in response

to the Boolean input Panel Show. The very large case structure on the right is only executed when the program is not running. Within this case, there are two major operations. The top case structure checks the Program VI State Cluster, down loads the motion program the first time the VI is executed, and parses the program for a program number and coordinate system. This is why you don't have to keep track of the program number of its CS. The large case structure below executes only when the panel is "Open and Active" and enables status monitoring, editing, and interactive execution from the panel.



To hide many nasty details from the user the VI maintains a Program VI State Cluster. When the VI executes the first time the Program Loaded item in the cluster is FALSE. Hence the program execution case on the left can't execute, the large case does, and the program is downloaded thereby updating the Program VI State Cluster indicating that the program is loaded, the program number, and associated coordinate system. At this point, the VI is knows everything it needs to run and monitor the program.

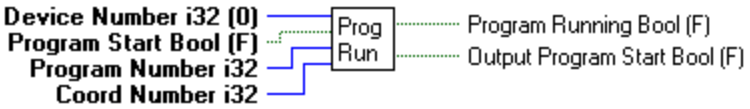
If your application does not need to be downloaded every time your system is turned on, changing the labeled Boolean constant on the top to TRUE with the mouse will disable automatic down load. This prevents the down load but doesn't provide the program number or coordinate system number. To provide this information set the constants in the case containing the Default Program # and Default CS # and save the VI.

After the down load is complete, repeated executions of this VI embedded in your application loop allow you to display the interactive panel or control and monitor the execution of your motion program. The VI provides its Program VI State Cluster as an output so you have access to this information for building tools that are more sophisticated.

If your application provides a latched TRUE input to Panel Show the panel is opened and activated. The sub-case within the large case can now execute and update the panel's display. This approach eliminates a fair amount of execution overhead in maintaining a panel when not displayed. The structure of this part of the diagram is obvious if you've followed the documentation so far. The one difference is that the Stop button has no loop to halt. Instead it closes the panel and allows continued execution of the other operations in the VI.

Once the program is loaded, the case on the far left can execute. This structure performs two sequential operations. First, the **PmacPQMArray** VI is executed. This will set specified PMAC P, Q, or M variables using Input PQM Variant Array when Program Run is TRUE. If Program Run is FALSE, the specified P, Q, or M variables are retrieved from PMAC and output in Output PQM Variant Array with New Output TRUE. The second operation is to monitor the associated motion program using **PmacProgRun**. If Program Run is TRUE the program is running and can use the newly loaded P, Q, or M variables. The Program Running output will be TRUE indicating that the program is running. If Program Run is FALSE no program is executing and Program Running will indicate whether another motion program is running in the associated coordinate system. **PmacProgRun** is covered in detail here.

- **PmacProgRun** - Control and monitor the execution of Program Number in Coord Number. The specified program is started when Program Start is TRUE and no program is currently running in Coord Number. Program Running indicates that some program - maybe not Program Number - is running in Coord Number. Output Program Start is a copy of Program Start and can be used to sequence program execution with other operations.



PmacPQMExamp

The encapsulation of a motion program with a wrapper is a huge step toward integrating PMAC with LabVIEW. The **PmacPQM** collection of VIs carry this further by providing an architecture for tying controls and indicators to the P, Q, and M variables used by your PMAC motion programs and PLCs. To illustrate how to do this we'll use **PmacPQMExamp** located in the directory **\PmacPQM**.

To begin lets look at the sample motion program **PmacPQMTest.pmc** that we want PMAC to run (**.pmc** is used by Delta Tau SW tools to indicate a motion program). You should note the associated encapsulation VI **PmacPQMTest.vi** created by **PmacTerminalEdit**.

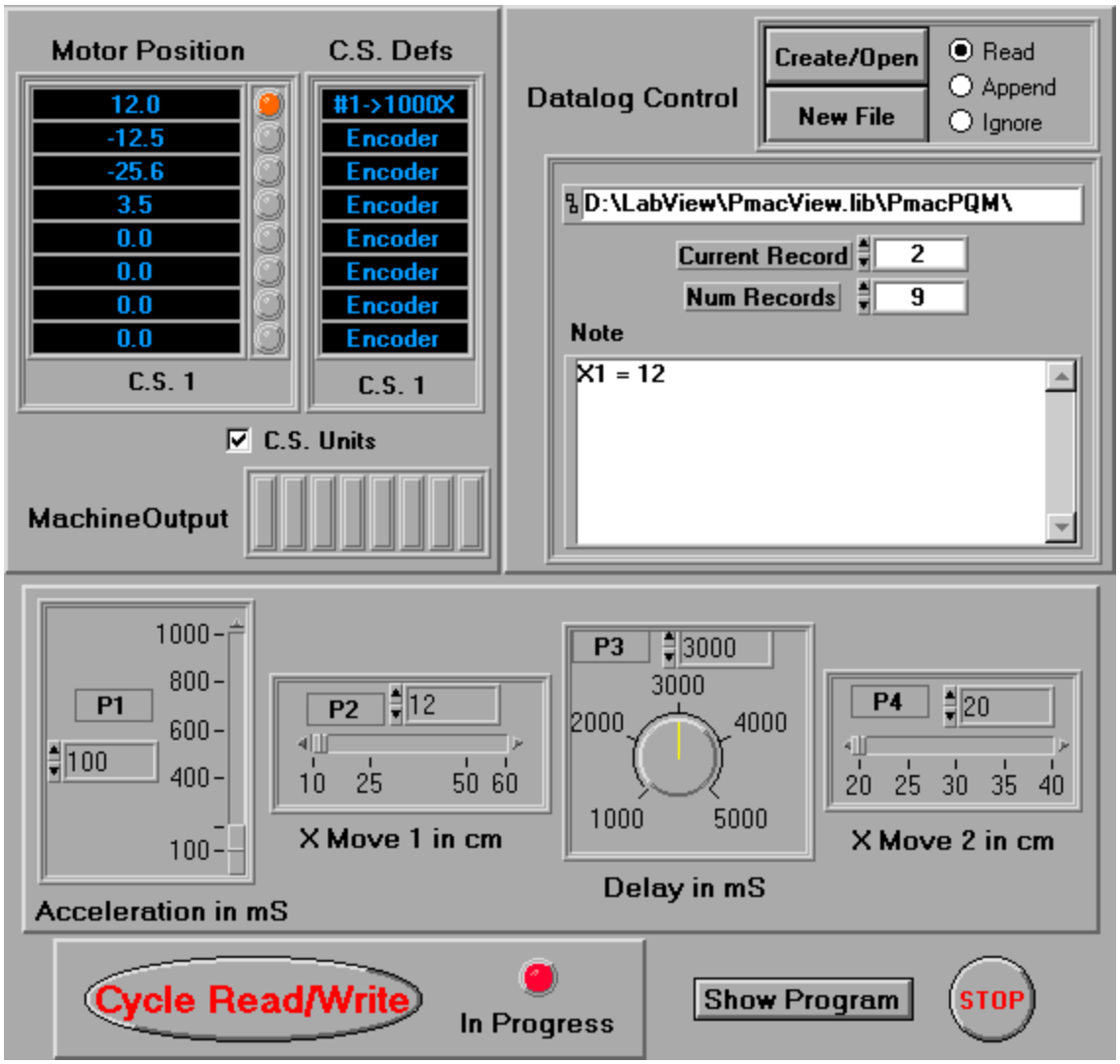
```
; PmacPQMTest.pmc

; USE CS &1 ; Parsed by PMACPanel during download
```

```
Close       ; Always close any open buffers
&1          ; Define the CS
#1->1000x

m1->*       ; Redefine M1 as standard output port
m1->y:$ffc2,8,8,u

open prog 32      ; Parsed during download
clear             ; Otherwise appended to buffer!

linear            ; Set move modes
abs

ta(P1)            ; Set move - Accel time is P1
ts250
tm1500

m1 == 1           ; Show bit on port move X to P2
X(P2)
DELAY(P3)         ; Delay for P3 mS

m1 == 2           ; Update the port
X(P4)             ; Move X to position P4

DELAY1500

ta250             ;  New move parameters
ts125
tm750

m1 == 4           ; Return home
x0
dwell 100

m1 == 0
close
```

This program uses four P-Variables to define its execution and motion.

- P1 - Acceleration time for first few moves
- P2 - First X position
- P3 - Delay time after move
- P4 - Second X position

Because P, Q, and M variables are used to configure a motion or PLC program PMACPanel provides a collection of VIs to take values from panel controls and set associated P, Q, and M variables for use by your programs. You can then start program execution. The panel for the example shows a familiar set of indicators to monitor motor motions on the top left.



Below this are four **PmacPQM** Cluster controls associated with the four P-Variables used by the program P1 - "Acceleration in mS", P2 - "X Move 1 in cm", etc. Each cluster contains a control for the value of the variable and a string control specifying which P, Q, or M variable. At the bottom of the panel is a Cycle Read/Write button to begin execution of the encapsulated program and an In Progress indicator to monitor the execution of the program. The Show Program button will open the encapsulated program's interactive panel thereby allowing you to interactively modify the program and step through its execution.

PmacPQM provides the ability to log PQM variables to standard LabVIEW datalog files. The logging process is controlled by the Datalog Control Cluster and Datalog Display Cluster in the upper right and is sequenced with the Cycle Read/Write button.

If you click the Create/Open button, you are prompted for the name of a datalog file. You can select an existing file created using this example or provide the name for a new file. There are two example files named **datalog.dat** and **datalog1.dat**. New File closes an existing log file and allows you to select a new one. This needs to be done prior to actually logging PQM data. The Read/Append/Ignore radio buttons define what to do with the PQM data when the Cycle Read/Write button is clicked.

- **Read** - It is assumed that you opened an existing data log file created earlier. Use the record specified by Current Record to read a PQM record, set the PQM variables in PMAC using the retrieved record, then execute the encapsulated motion program. You will see the values on the cluster controls change to those read from the record when Cycle Read/Write is clicked. Using this you can replay previously executed tests and configurations. The state of the panel illustrates that a Read operation was performed during the last cycle using record 1 (after the cycle Current Record was incremented to 2). The note indicates that X1 = 12 and indeed P2 has a value of 12.

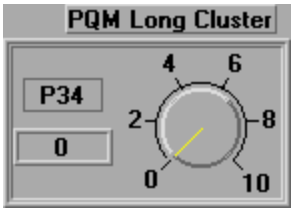
- **Append** - Read the PQM cluster controls, append them to the datalog file, send them to PMAC, and start the execution of the encapsulated motion program. You can add a note to the record prior to clicking the Cycle Read/Write button.

- **Ignore** - Keep the datalog file but do not read or write anything. Simply pass the PmacPQM clusters to PMAC.

PmacPQM Clusters

There are four standard **PmacPQM** clusters provided for use on your applications panels. Defining these clusters binds the PQM variable's name with the actual numerical value to be used with the variable. These are based on variations of the cluster definition for **PmacPQMLong**.

- **PmacPQMLong** - Cluster for tying PQM variable definition with an i32 control/indicator. After inserting on your panel specify a PQM variable name for the Variable Item and make it the default using **Right Mouse Button»Data Operations»Make Current Value Default**. Replace Control to reflect your requirements.



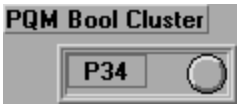
PQM Long Cluster Cluster for tying PQM variable definition with an i32 control/indicator. After inserting on your panel specify a PQM variable name for the Variable Item and make it the default using **RightMouseButton»Data Operations»Make Current Value Default**. Replace Control to reflect your requirements.

Variable String defining PQM Variable name. e.g. "P34"

Control Control for associated PQM Variable

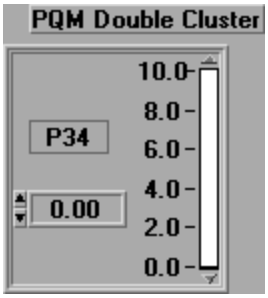
PQM Type

- **PmacPQMBool**



- **PmacPQMShort**

- **PmacPQMDbl**



When you insert these on your panel feel free to move the items around, replace the actual control, change the format and range, color, Boolean text, etc. Remember to keep the cluster order as indicated. When you define the name of the Variable item in the cluster, it is a string (i.e. P34). You need to set this as the default for each control in your panel and save the VI using the cluster – not the original cluster itself!

PmacPQMVariant functions as a neutral or void type of PQM cluster.

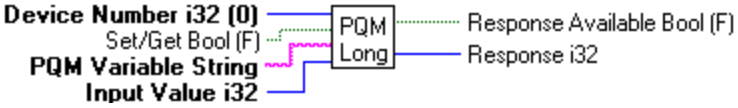
- **PmacPQMVariant** - Cluster for tying PQM variable definition with a PQM type-neutral string. This cluster is generally not used on application panels.

PmacPQM Conversions

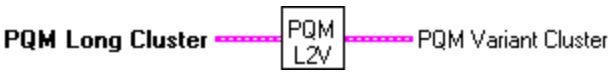
There are three types of PQM VI for processing PQM clusters. The examples given here are for the **PmacPQMLong** collection. Similar collections exist for **PmacPQMBool**, **PmacPQMShort**, and **PmacPQMDbl**.

- **PmacPQMLong** - If Set\Get is FALSE or not wired get the Long PQM Variable specified by PQM Variable String. Response Available will be TRUE to indicate Response contains the new value. If Set\Get is TRUE set the Long PQM Variable using Input Value. Response Available will be FALSE and Response defaults to Input Value.

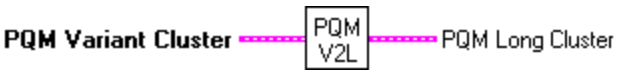
If you specify an M-Variable it must be defined using Pewin32, PmacTerminal, or PmacCommSendString.



- **PmacPQMLong2Var** - Convert the PQM Long Cluster to a type-neutral PQM Variant Cluster.

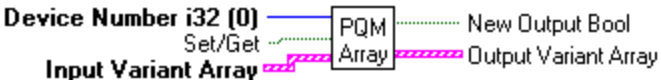


- **PmacPQMVar2Long** - Convert a type-neutral PQM Variant Cluster to a PQM Long Cluster.



The purpose of the 2Var and Var2 VIs is to convert clusters of specific types to and from neutral **PmacPQMVariant** types for building arrays that bundle PQM cluster controls into a single item.

- **PmacPQMArray** - Set or Get a collection PQM variables as defined by an array of PmacPQMVariant clusters.



The use of arrays greatly simplifies the development of PQM configuration panels for your applications. **PmacProgSubVI** VIs created by **PmacTerminalEdit** accepts the arrays as inputs and provide them as outputs. This allows you to update program PQM variables prior to actually executing the program and monitor any PQM variable used by the program as it executes.

PmacPQM Datalogging

PMACPanel supports data logging of **PmacPQM** clusters using the VIs in the **PmacFile** collection. These can be modified to support record attributes such as time stamps in support of your particular needs.

- **PmacFileDatalog** - Manage datalog operations for type-neutral PmacPQMVariant Arrays.

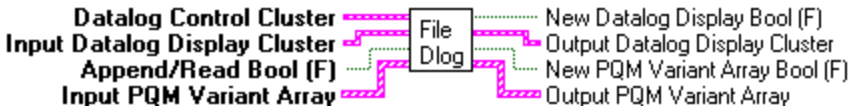
  Operations as specified by the radio buttons in Datalog Control Cluster are performed when Append/Read is TRUE. A file must be selected prior to executing the operation using the Create/Open button or New File button in the cluster. The file is opened and closed on every transaction. After an operation New Datalog Display is TRUE and Output Datalog Display Cluster contains updated operation status for your application's cluster.

  Append operations write Input PQM Variant Array to the end of the file specified in Input Datalog Display Cluster and update Current Record and

Num Records in the output cluster. The contents of the Note window are appended with the record.

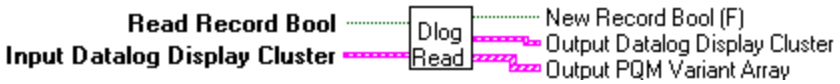
Read operations read the record specified by Current Record in Input Datalog Display Cluster from the specified file and generate a new Output PQM Variant Array. The availability of new data is indicated by New PQM Variant Array TRUE. Output Datalog Display Cluster increments Current Record and displays the Note, if any, attached to the record. Read operations cannot read past the end of the file and simply read the last record in the file.

To change the data logged by this VI simply change Input and Output PQM Variant Array to your own data type. Similar modifications to PmacFileDatalogAppend, Create, and Read are also required.



- **PmacFileDatalogAppend** - When Append Record is TRUE append Input PQM Variant Array to the file specified in Input Datalog Display at the end of the file. Update the Current Record and Num Records in Output Datalog Display Cluster. Indicate the new data by setting new Datalog Display TRUE.

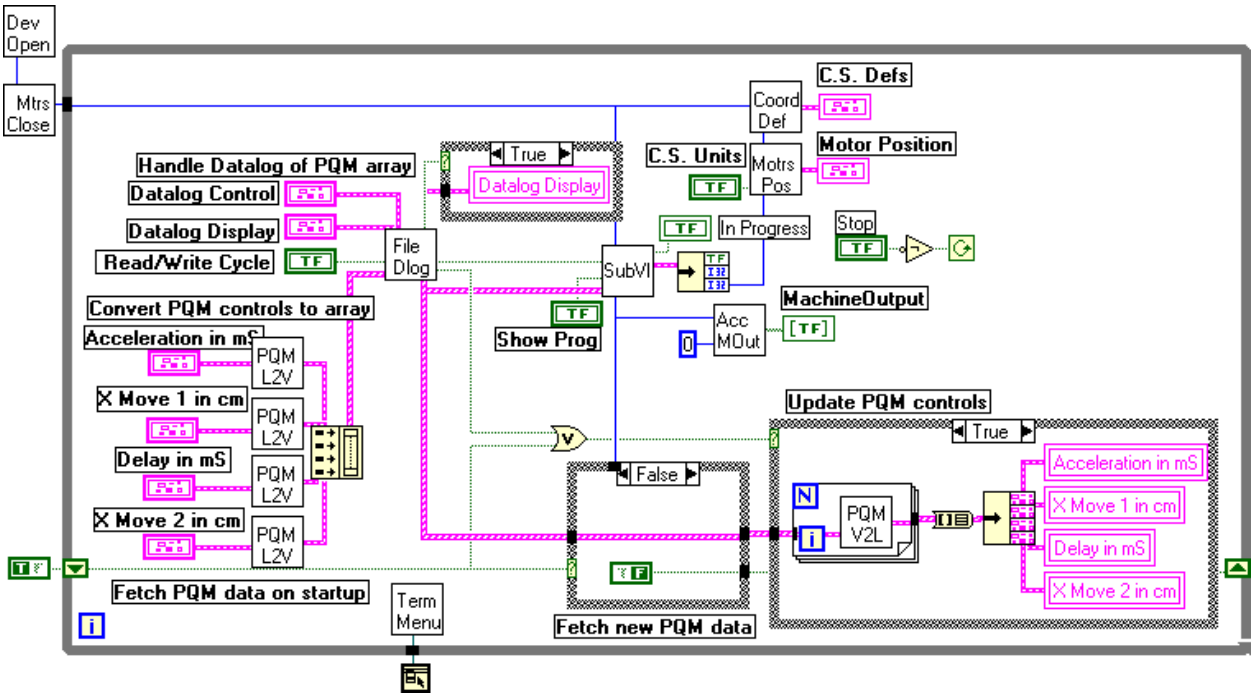
- **PmacFileDatalogRead** - When Read Record is TRUE read Output PQM Variant Array from the file specified in Input Datalog Display using Current Record. Update the increment Current Record in Output Datalog Display Cluster and display the Note, if any, stored with the record. Indicate the new data by setting new Datalog Display TRUE.



- **PmacFileDatalogCreate** - Create or Open an existing datalog file to store data of type Input PQM Variant Array along with notes. When Create/Open is TRUE use the path in Input Datalog Cluster. If this path is empty or Not A Path display a file selection dialog. When a file name is entered or an existing file is selected the number of records in the file is determined. All updated information is available in Output Datalog Display Cluster and indicated by New Datalog Display TRUE.

Using Encapsulated Motion Programs

We've already seen the panel for **PmacPQMExamp**. Lets look at how **PmacPQM** ICVs can be combined with the custom **PmacProgSubVI** created by **PmacTerminalEdit** to build great applications. The diagram for the example is shown here.



The application has the standard execution loop with menu processing and a Stop button. In the middle is the **PmacProgSubVI** VI created for the motion program **PmacPQMTest.pmc** already introduced. When the Cycle Read/Write button is clicked a sequence of operations begins. The **PmacPQM** clusters on the panel are translated into **PmacPQMVariant** clusters and bundled into an array. The array is passed to **PmacFileDatalog,** which appends the array to the datalog file, ignores data logging, or ignores the current input and reads a record from the specified data log file. The array is passed to the encapsulated motion program VI along with a TRUE Boolean. The wrapper VI will down load the variables to PMAC and start the program.

If **PmacFileDatalog** has new PQM array data due to because it read the data from a datalog file or simply passed the input array through, the PQM clusters are updated with the PQM array. This is done by the two case structures in the lower right of the diagram that convert the array items to appropriate types, unbundle them, and set the local variables for the clusters on the panel. If this is the first execution of the VI the shift register will query PMAC for the current PQM variables, and update the clusters.

This example program indicates its location in the program by setting bits of a standard memory mapped machine output. The output is monitored by **PmacAccMachineOutput8** and used to drive an indicator on the panel. The VIs and indicators in the upper right display the coordinate system definitions and motor position. The coordinate system number for the VIs is obtained from the encapsulated motion program VI.

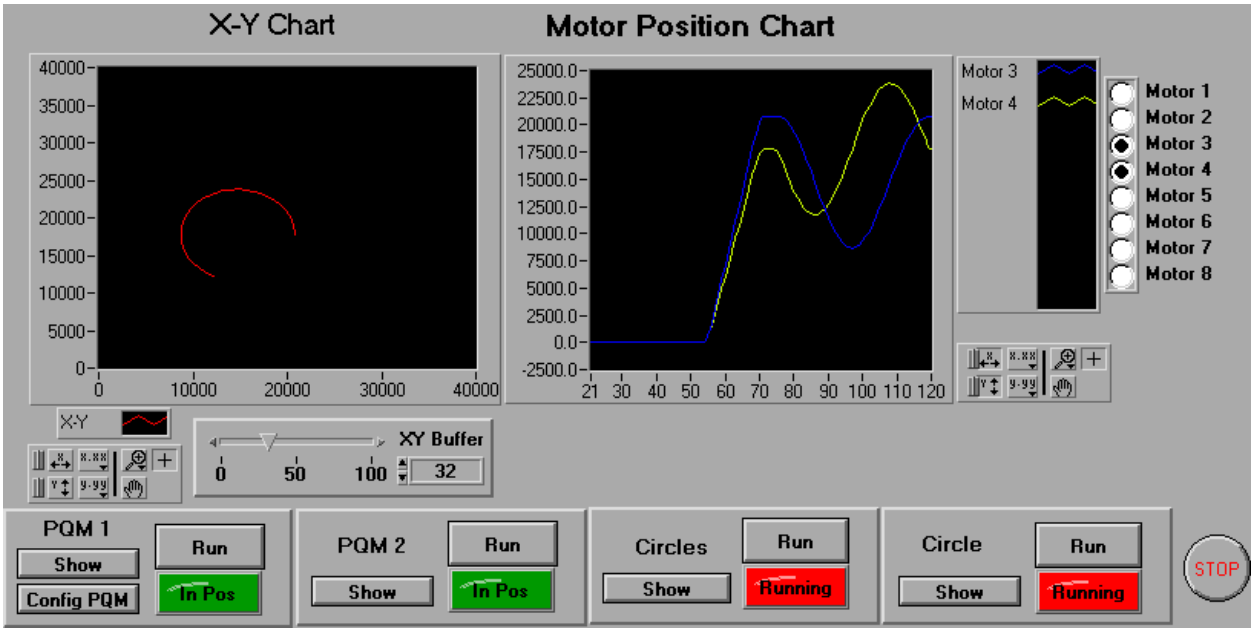
PmacTestExamp

Development of your interactive application framework can get involved. If you've followed everything up to this point, you understand how PMACPanel cooperates with PMAC, how to use the various ICV's in your applications, and how to encapsulate motion programs using PMACPanel. This section discusses one framework for controlling and configuring multiple motion programs.

The example **PmacTestExamp**, located in the **PmacTest** directory, has four encapsulated motion programs.

- **PmacTestPQM1.pmc** in CS 1
- **PmacTestPQM2.pmc** in CS 2
- **PmacTestCircle.pmc** in CS 3
- **PmacTestCircles.pmc** in CS 3

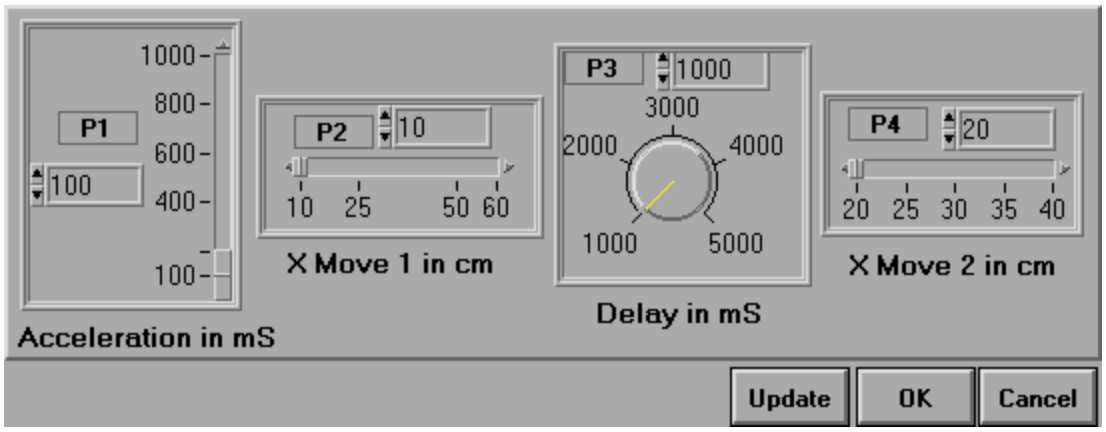
PmacProgSubVI's were created for each program by running **PmacTerminalEdit**, loading the programs one at a time, and clicking the Create Prog VI. This takes 2 minutes to do. **PmacTestPQM1** and **PmacTestPQM2** are similar to **PmacPQMExamp** and use P-Variables to configure their motion.



The panel for **PmacTestExamp** is shown above. The panel shows four sets of controls – one for each program.

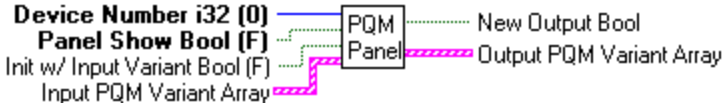
- A button to start the program
- A button to show the encapsulated motion program's execution panel
- An LED to indicate the execution state of the program

PQM 1 has an extra button that, when clicked, allows modification of its associated P-Variables with the control panel shown below. The VI architecture for doing this isn't really a PMACPanel design issue, but it demonstrates an approach for PQM configuration using pop-up panels in a larger application.

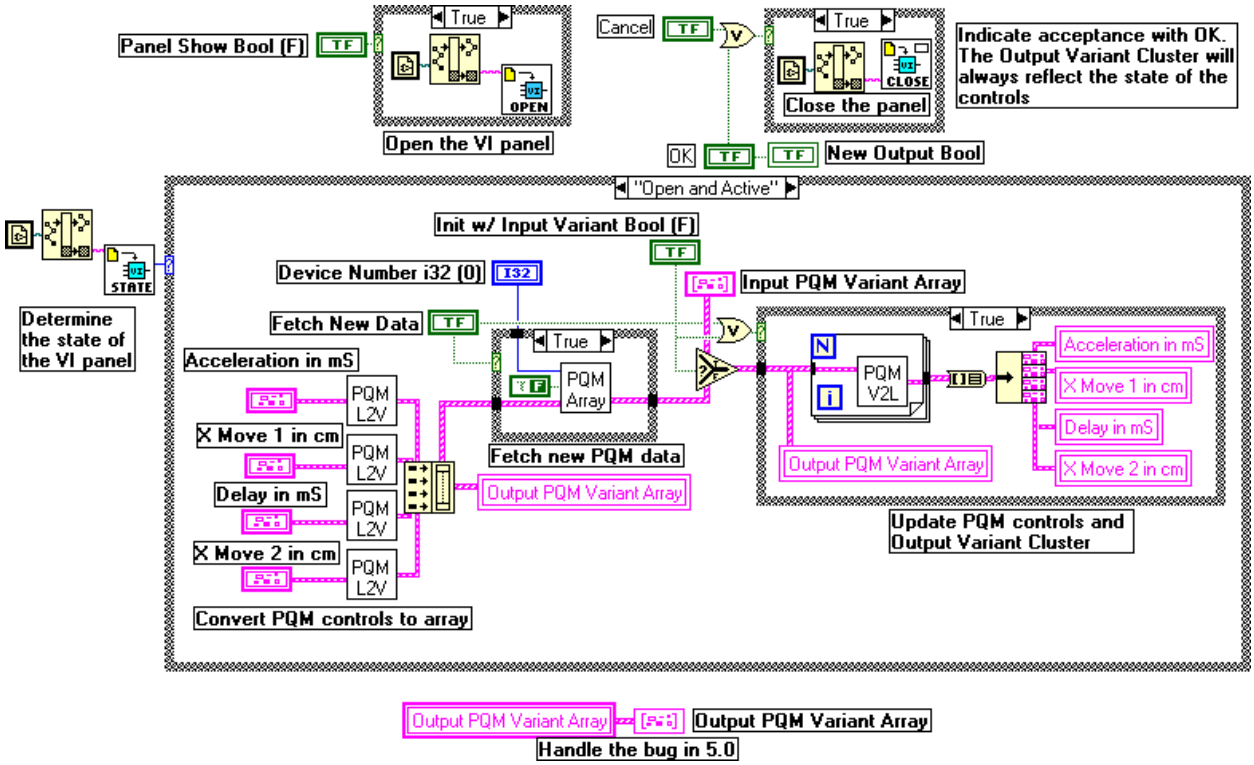


This application specific PQM configuration VI doesn't actually send the variables to PMAC. It creates a **PmacPQMVariant** array from the panel clusters that can be used by the encapsulated motion program VI in the main application. Update reads the current P-Variable values from PMAC and updates the controls on the panel. You can close the panel using Cancel and discard the new control values. If you click OK, the VI indicates there is a new PQM Variant Array available. The disposition of the new data is up to the main application VI. The description for the VI is given here.

- **PmacTestPQM1Panel** - Group several PQM clusters together and coordinate their operation with an encapsulated motion program VI. Panel Show TRUE displays the panel. If you supply Input PQM Variant Array and set Init w/ Input Variant Array TRUE the controls initialize themselves using the array contents when the panel is displayed. If you do not use these inputs you should first Update the controls from PMAC. Output PQM Variant Array maintains any changes made using the controls from execution to execution. If the user clicks OK New Output Bool reflects this. Otherwise Output PQM Variant Array contains the current state of the cluster controls.

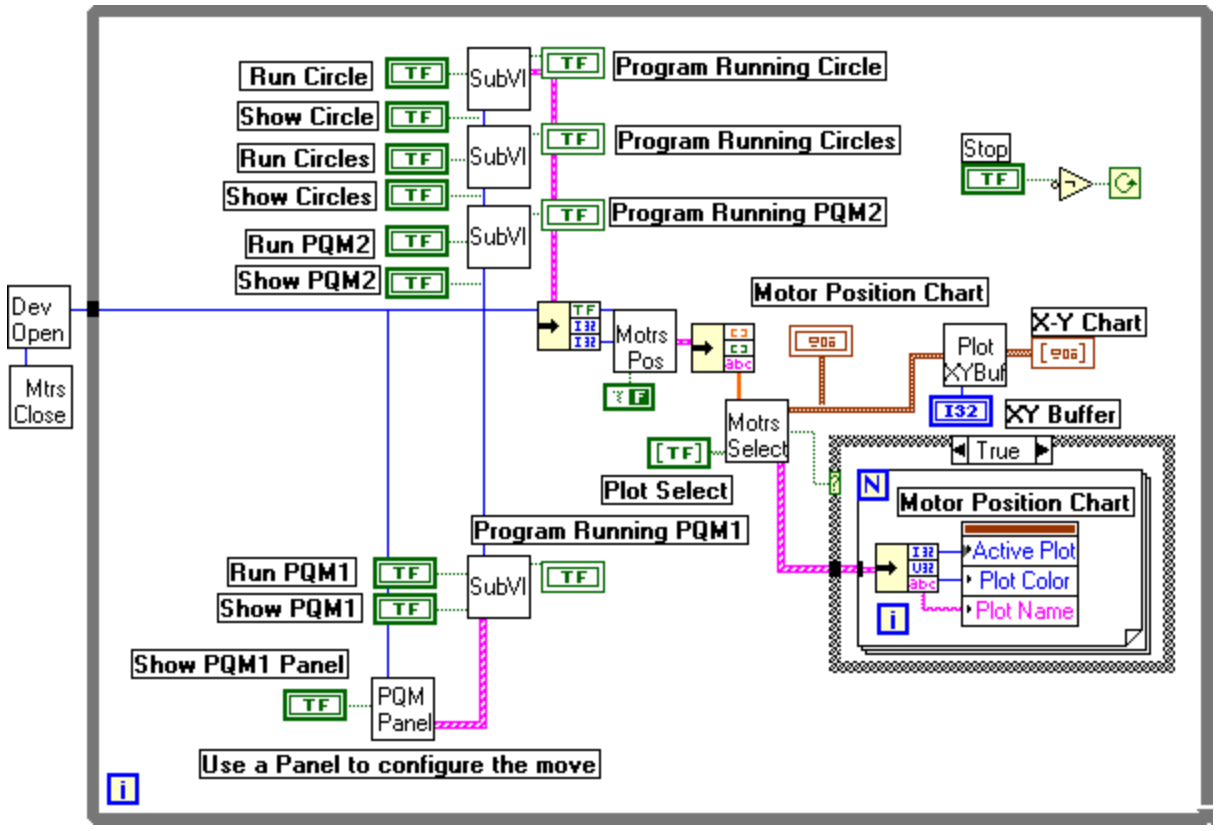


The VI forms a basis for generating your own pop-up PQM panels. The diagram is shown here and has pieces of **PmacProgSubVI** and **PmacPQMExamp** in it.



As with **PmacProgSubVI,** there is no execution loop. If Panel Show is TRUE the panel is opened by the case structure on the top left. If the panel is "Open and Active" the large case structure is enabled and the controls on the panel are active. The panel clusters are converted to variants and used to set Output PQM Variant Array. Thus, Output PQM Variant Array always reflects the current state of the controls. If Init w/Input Variant (latched) is TRUE or if the user clicks Update the controls are updated using either the newly fetched data or the Input PQM Variant Array. When Cancel or OK is clicked the panel is closed. The state of the OK button is used to set New Output.

Incorporation of this pop-up VI into **PmacTestExamp** can be seen in the example's diagram.



The handling of the Run, Show Panel, and In-Progress indicators for the individual motion programs is very simple because of the encapsulation VIs. At the bottom, the Show PQM1 Panel button is supplied to **PmacTestPQM1Panel** to allow configuration of the PQM Variant Array supplied to the appropriate encapsulation VI.

The implementation of the plots, charts, and other indicators is identical to that covered already.

Chapter 7 - Homing, Encoders, and Position Capture

Basics

PMAC utilizes a custom gate array to interface motor encoders to PMAC and perform a number of high-speed computations required to monitor motor position. When writing PMAC programs you specify moves in coordinate system units. Motor positions are specified in encoder counts. The gate array uses another version of encoder counts to translate motor position into encoder position.

In Chapter 4 we introduced the **PmacMotor** and **PmacCoord** collection of ICVs that allow you to convert between motor position in encoder counts and motor position in coordinate system units. In this chapter we complete the picture by introducing the **PmacEncoder** and **PmacHome** collection of ICVs that give you the ability to move freely between coordinate system, motor position, and encoder position specifications. These are important if you want to relate precise position information to actions in your system. Using the encoder gate array, you can configure PMAC to

- Capture positions in response to external or internal triggers
- Generate triggers at pre-specified compare-equal encoder positions

The first operation required for precision position measurement of any sort is the establishment of a zero or home position. On PMAC this is done using an encoder capture operation that is triggered by a home position sensor. Homing details are covered in detail in the *PMAC User Manual*. Some of this information is repeated in this chapter.

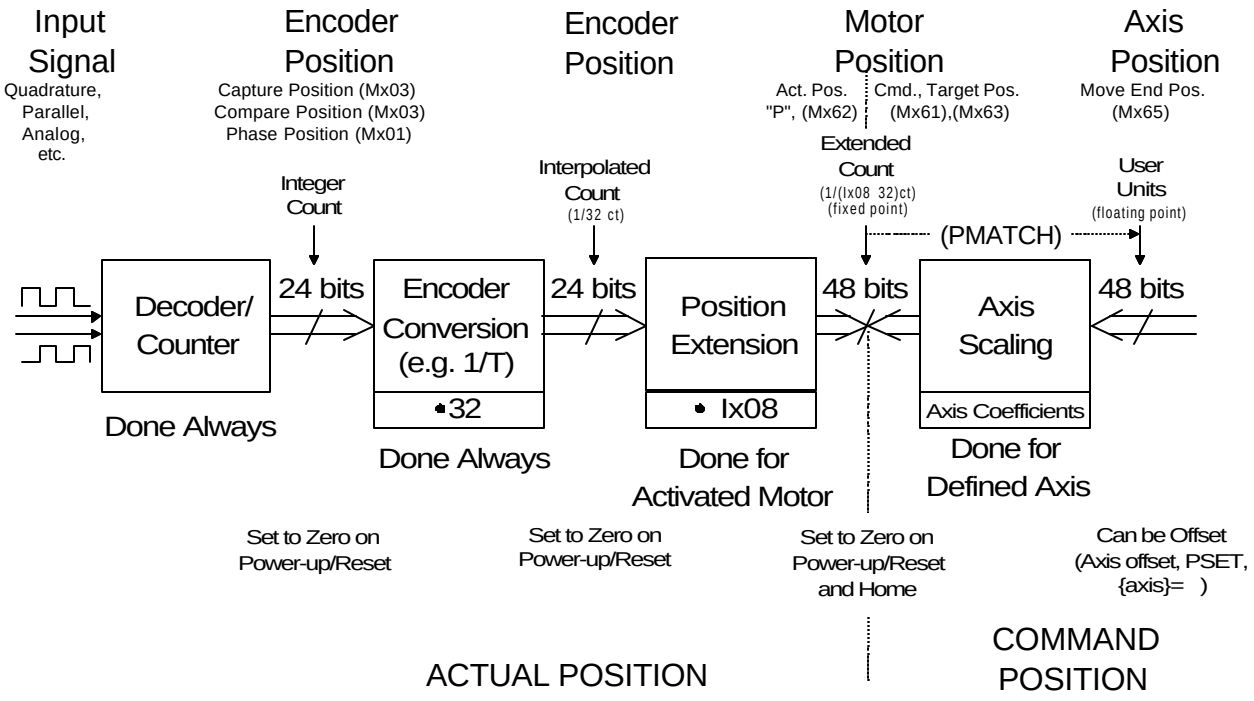
In this Chapter we cover homing and position capture operations. In the next chapter we will demonstrate how this same capability can be used to capture motor positions in response to external events generated by another National Instruments DAQ system or instrument. Compare operations will also be covered in the next chapter and allow you to precisely synchronize data acquisition with motion.

Position Basics

As shown below, PMAC takes position information from a 24-bit encoder register pointed to by Ix03 and extends it in software to a 48-bit register for the actual motor position. In the process of extension, it multiplies the encoder value by the position scale factor Ix08. Because the register in the encoder conversion table is in units of 1/32 of a count, the actual motor position register is in units of 1/(Ix08*32) of a count.

The extended motor position registers are set to zero on power-up and reset (unless there is an absolute position sensor), and again at the end of a homing search move. The encoder position registers are only set to zero on power-up and reset. Therefore, after a motor is homed, there is an offset between a motor's zero position and its encoder's zero position.

You must understand this offset because you will be using the encoder registers for position capture and compare not the motor registers. Depending on your mechanical configuration, you may also have to handle the rollover of encoder registers if they will be traveling more than the +/-8 million counts supported by the 24-bit encoder register. The modulo (%) operator is useful for this. For more details, refer to *Synchronizing PMAC to External Events* in the *PMAC User Manual*.



Motor position is always kept in terms of encoder counts. When a motor is assigned to an axis through a Coordinate Definition statement as in

```
&1
#1->1000X
```

for use in a motion program, the scale factor in the statement determines the units of the axis (usually inches, millimeters, degrees, etc.). As introduced in Chapter 4, programmed moves for an axis are converted to motor positions using the scale factors from the Coordinate Definition statements. It is important to realize that this conversion is for *commanded* positions only, and that the conversion normally goes only one way: from *axis* to *motor. PMAC never computes actual axis positions*

Position-Capture

PMAC's position-capture function latches the current encoder position into a special register at the time of an external trigger. The operation is set up, and later serviced, in software. The actual latching is executed in hardware, without the need for software intervention. This means that the only delays in a position capture are the hardware gate delays (less than 100 nsec) thereby providing a very accurate capture function.

Trigger Condition

The position capture register can be used both "automatically", as in the firmware homing routines that handle the register directly, and "manually", where your program handles the register. Manual handling of the capture register will be covered in Chapter 8.

During motor setup, Ix25 specifies which set of flags (associated with one of the encoder counters) is used for that motor. It is important that the flag number match the position encoder number for the motor. If you use ENC1 for position-loop feedback, you should use Flags1 (HMFL1, +/-LIM1, FAULT1), and CHC1 as the encoder index channel.

The trigger event that causes the position capture is determined by Encoder I-Variables 2 and 3 (I902 and I903 for Encoder 1). Encoder I-Variable 2 defines what combination of encoder third-channel (CHC Index channel) transition and encoder flag transition trigger the capture. If Encoder I-Variable 2 specifies the use of a flag, Encoder I-variable 3 determines which flag (usually the home flag HMFLn). Once these have been configured, the on-line HOME command will use the position-capture feature automatically.

Homing

Homing is a PMAC firmware function that automatically performs a number of operations to establish a motor's zero position. The homing search move can be executed with the on-line **HOME** command, from a PLC program using **COMMAND"HOME"**, or a motion program **HOME** statement. However the **HOME** command is issued, Ix23 specifies the move's speed *and direction*. If Ix23 is greater than zero, the homing search move will be positive. If it is less than zero the move is negative. The acceleration for a homing search move is controlled by the same parameters -- Ix19, Ix20, and Ix21 -- as jogging moves

Action on Trigger

During the homing search move, PMAC firmware waits for the hardware trigger. When the trigger occurs PMAC reads the position at the time of capture, usually the hardware capture register, and uses it and the Ix26 home offset parameter to compute the associated motor's new encoder zero position.

Motor positions will now be referenced to a new encoder zero position plus or minus any axis offset in the axis definition statement. If the axis definition is

```
#1->10000X+3000
```

the home position will be reported as 3000 counts.

If software over travel limits are used (Ix13, Ix14 not equal to zero), they are re-enabled at this time after having been disabled during the search for the trigger. The trajectory to the new zero position is calculated using deceleration and reversal if necessary. Note that if a software limit is too close to zero, the motor may not be able to stop and reverse before it hits the limit. The motor will stop under position control with its commanded position equal to the home position. If there is a following error, the actual position will be different by the amount of the following error.

Home Complete

If you are monitoring the motor from a PLC program or PMACPanel to see if it has finished the homing move, it is best to look at the "home complete" and "desired velocity zero" motor status bits. The "home complete" bit is set to FALSE on power-up and reset; it is also set to FALSE at the beginning of a homing search move, even if a previous homing search move was completed successfully. It is set to TRUE as soon as the trigger is found in a homing search move, before the motor has come to a stop.

The "home search in progress" bit is simply the inverse of the "home complete" bit during the move: it is TRUE until the trigger is found, then FALSE immediately after. Therefore the monitoring should also look for the "desired velocity zero" status bit to become TRUE, which will indicate the end of the move.

Home Position Offset

Prior to V1.14 firmware, this value could be obtained by using the PLC program **HOMOFFST.PMC**, shown in the Examples section of the *PMAC User Manual*. Starting in V1.14, PMAC stores this value automatically.

PMAC automatically stores the encoder position captured during the homing search move for the motor. This value is kept in the Motor Encoder Position Offset Register [Y:$0815 (Motor 1), Y:$08D5 (Motor 2), etc.], which is set to zero on power-up/reset for motors without absolute power-on positioning. If Ix10>0 to specify an absolute power-on position read from a resolver so no homing is necessary, this register holds the negative of the power-on resolver position. In either case, it contains the difference between the encoder-counter zero position (power-on position) and the motor zero (home) position, scaled in counts.

There are two main uses for this register. First, it provides a reference for using the encoder position-capture and position-compare registers. These registers are referenced to the encoder zero position, which is the power-up position, not the home (motor zero) position. This register holds the difference between the two positions. This value should be subtracted from encoder position (usually from position capture) to get motor position, or added to motor position to get encoder position (usually for position compare).

To move an axis until a trigger is found, then convert the captured encoder position to a motor position, you can use the following M-variable definitions:

```
M103->X:$C003,24,S  ; ENC1 position-capture register
M117->X:$C000,17    ; ENC1 position-capture flag
M125->Y:$0815,24,S  ; #1 encoder pos offset register
```

Zero-Move Homing

If you have following error when you give the **HOMEZ** command, the reported actual position after the **HOMEZ** command will not be exactly zero; it will be equal to the negative of the following error.

If you wish to declare your current position the home position without commanding any movement, you can use the **HOMEZ** (on-line) or **HOMEZn** (motion program) command. These are like the **HOME** command, except that they immediately take the current commanded position as the home position. The Ix26 offset is not used with the **HOMEZ** command. This is not a reliable home and the PMACPanel VIs introduced in this chapter and the next do not handle this phantom home offset. You can, if desired, fake this by modifying **PmacEncoderOffset**.

Homing Into a Limit Switch

The polarity of the limit switches is the opposite of what many people would expect. The -LIMn input should be connected to the limit switch at the *positive* end of travel; the +LIMn input should be connected to the limit switch at the *negative* end of travel.

It is possible to use a limit switch as a home switch. However, you must first disable the limit function of the limit switch if you want the move to finish normally; if you do not do this, the limit function will abort the homing search move. Even so, the home position has been set; a J=0 command can then be used to move the motor to the home position.

To disable the limit function of the switch, you must set bit 17 of variable Ix25 for the motor to 1. For example if I125 is normally $C000 (the default), specifying the use of +/-LIM1 for motor 1, setting I125 to $2C000 disables the limit function.

It is a good idea to use the home offset parameter Ix26 to bring your home position out of the limit switch, so you can re-enable the limits immediately after the homing search move, without being in the limit.

Homing from PLC and Motion Programs

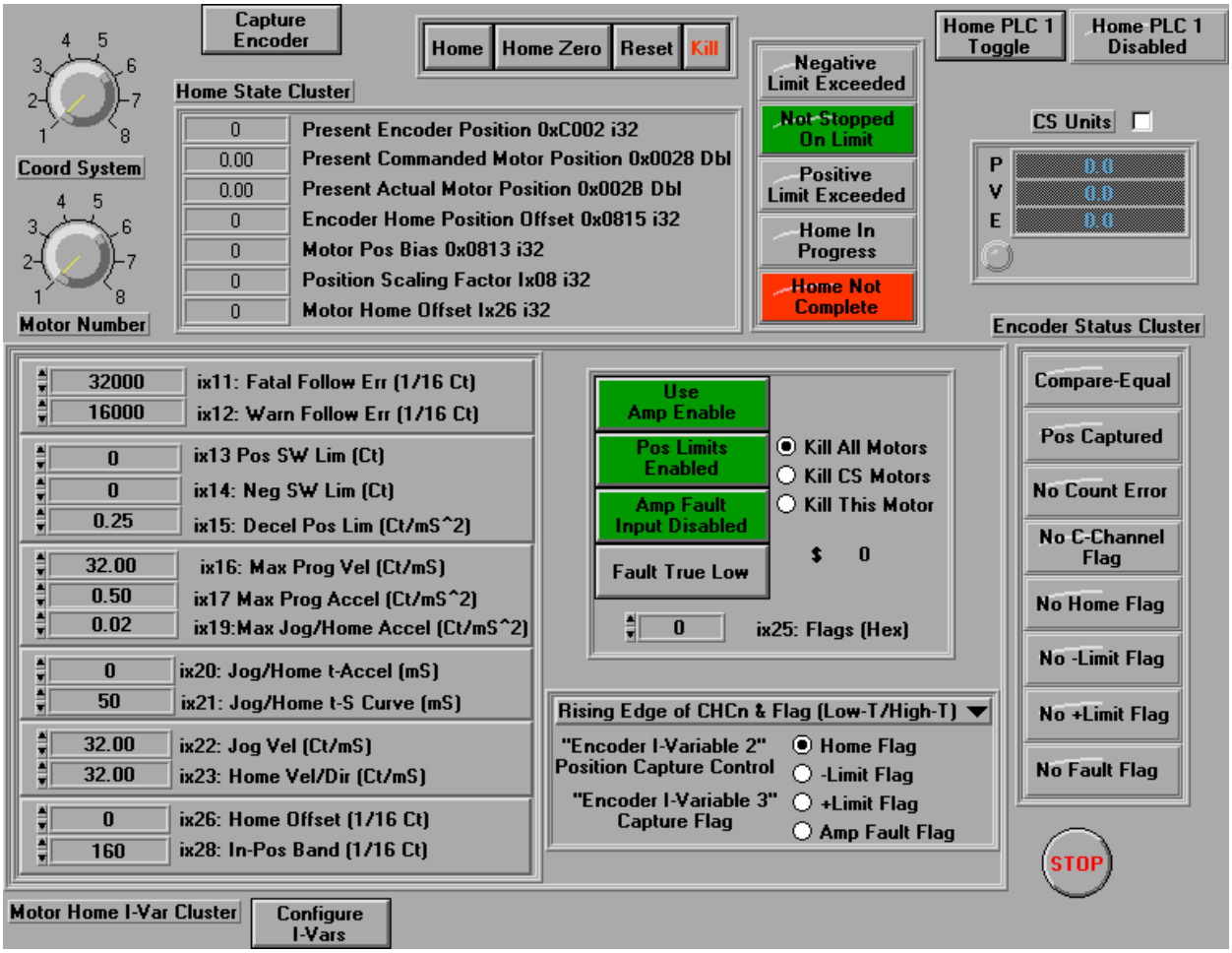
The *PMAC User Manual* has an extensive section on homing techniques using PLC and motion programs. These are not covered in this manual. However, the programs for these are included in the **PmacHome** collection of ICVs.

PmacHomeExamp

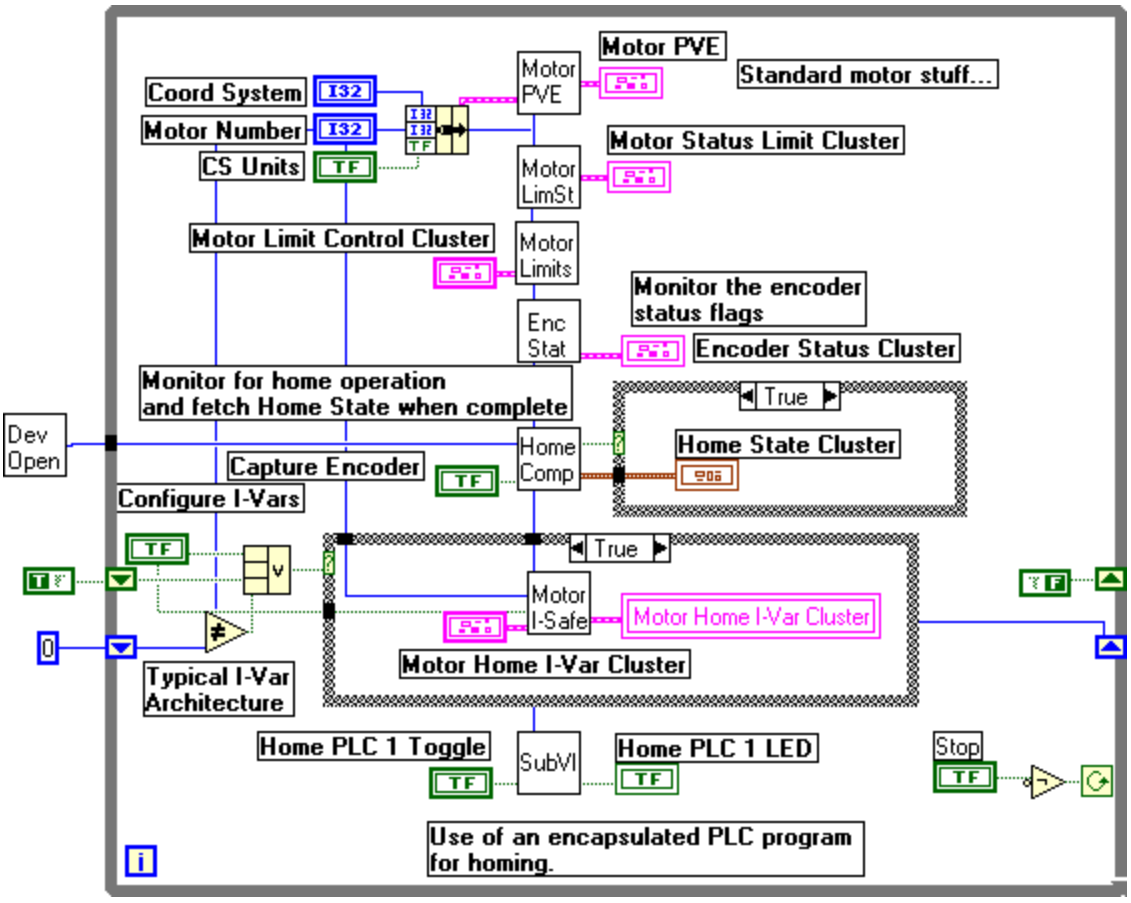
Having covered the basics of position capture and homing from a purely PMAC perspective we can now look at the ICVs available for use in your applications. We'll start by examining the panel for **PmacHomeExamp** shown below.

Many of the panel clusters should look familiar. There are three new indicator clusters associated with homing and a few new ideas associated with PLC program encapsulation. In the bottom left is a very large **PmacHomeIVar** cluster that borrows extensively from **PmacMotorIVarSafety** and **PmacMotorIVarMove**. It adds a new cluster for Encoder I-Vars 2 and 3. On the far right is an indicator bar that directly displays eight encoder status bits. Next to the Motor Number and Coord System knobs is a Home State Cluster. This contains data from several I-Variables and memory registers that define how motor position is transformed to encoder position. The cluster is updated any time the Capture Encoder button is clicked. In the top right is a button that

will toggle the Execution State of the encapsulated PLC program that sets up and executes a homing operation.



Before going into the individual pieces of this example lets look at the diagram below. As usual, the execution loop has several motor ICVs and a standard I-Variable architecture. The **PmacEncoderStat** VI and cluster monitor the encoder status bits and **PmacHomeComplete** monitors the execution of homing moves and retrieves the Home State Cluster. At the very bottom is the encapsulated PLC program Sub VI.



Configuring the Position Capture Trigger

Earlier we discussed position capture and homing. Homing is a firmware operation that uses the encoder hardware position capture capabilities to establish the relationship between a motor's zero position and the encoder's zero position. To perform a homing operation three things are required

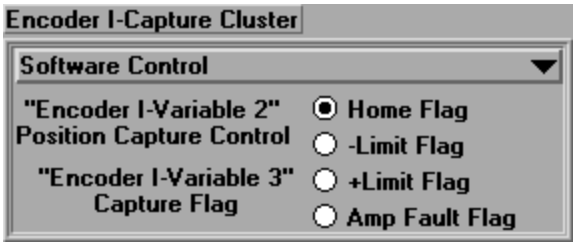
1. One of PMAC's four HW trigger flags HOME, -LIM, +LIM, or FAULT must be selected.

2. A trigger condition specifying the rising or falling edge of the flag possibly combined with the encoder index channel C must be selected.

3. If a FAULT or +/-LIM flag is used the limit or amplifier disable capabilities must be disabled. If you are using these PMACPanel can perform the necessary steps, but you must consult the *PMAC User Manual* for details so that you really understand what you are doing.

The **PmacHomeIVar** cluster on the example panel handles the configuration of Ix25 and encoder I-Variables 2 and 3. **PmacMotorIVarFlag** controlling Ix25, is detailed in Chapter 4. The **PmacEncoderIVarCapture** cluster and its two sub controls are documented here.

- **PmacEncoderIVarCapture**



"Encoder I-Variable 2" Position Capture Control This parameter determines which signal or combination of signals (and which polarity) triggers a position capture of the counter for encoder n. If a flag input (home, limit, or fault) is used, I903 (etc.) determines which flag. Proper setup of this variable is essential for a successful home search, which depends on the position-capture function. The following settings may be used:

Setting	Meaning
0	Software Control
1	Rising edge of CHCn (third channel)
2	Rising edge of Flag n (as set by Flag Select)
3	Rising edge of [CHCn AND Flag n]
4	Software Control
5	Falling edge of CHCn (third channel)
6	Rising edge of Flag n (as set by Flag Select)
7	Rising edge of [CHCn/ AND Flag n]
8	Software Control
9	Rising edge of CHCn (third channel)
10	Falling edge of Flag n (as set by Flag Select)
11	Rising edge of [CHCn AND Flag n/]
12	Software Control
13	Falling edge of CHCn (third channel)
14	Falling edge of Flag n (as set by Flag Select)
15	Rising edge of [CHCn/ AND Flag n/]

Note that several of these values are redundant. To do a software-controlled position capture, preset this parameter to 0 or 4; when the parameter is then changed to 8 or 12, the capture is triggered (this is not of much practical use).

Encoder I-Variable 3" Capture Flag This parameter determines which of the "Flag" inputs will be used for position capture (if one is used -- see I902 etc.):

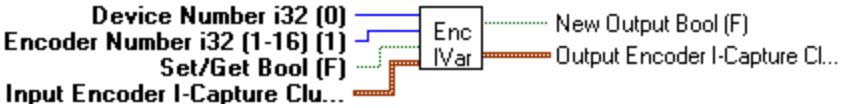
Setting	Meaning
0	HMFLn (Home Flag n)
1	-LIMn (Negative Limit Signal n)
2	+LIMn (Positive Limit Signal n)

3 FAULTn (Amplifier Fault Signal n)

This parameter is typically set to zero, because in actual use, the +/-LIMn and FAULTn flags create other effects that usually interfere with what is trying to be accomplished by the position capture. If you wish to capture on the +/-LIMn or FAULTn flags, you must either disable their normal functions with Ix25, or use a channel n where none of the flags is used for the normal axis functions.

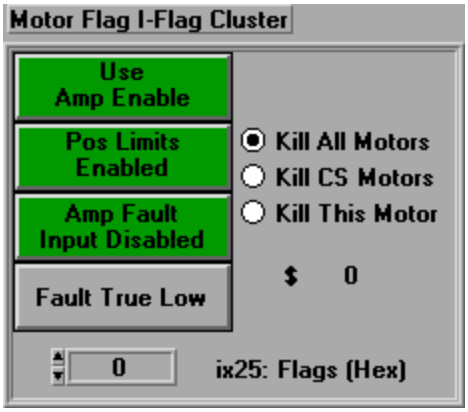
The VI for the cluster is

- **PmacEncoderIVarCapture** - Follow PMACPanel's standard I-Variable VI architecture. When Set/Get is TRUE the Encoder I-Variables for the specified Encoder Number are set. Otherwise they are fetched from PMAC and provided by Output Encoder I-Capture Cluster with New Output TRUE. Set/Get is not required and defaults to a Get operation.



As noted if you use the +/-LIM or FAULT flags steps must be taken to modify their normal operation. These are safety limits that as a rule stop motion and disable the amplifier – not useful when you are trying to home the motor or set you limits. You can modify this behavior using the **PmacMotorIVarFlag** cluster covered in Chapter 4.

- **PmacMotorIVarFlag**



By disabling the Position Limits and/or Amp Enable you can home into a +/-LIM or FAULT flag. Make absolutely certain you have read the *PMAC User Manual* section on these topics and understand what you are doing. As an example of potential problems, consider this. When homing into a Limit switch you must start the move on the proper side of the switch and move toward it. Otherwise, you will move away from the switch and might hit a mechanical stop.

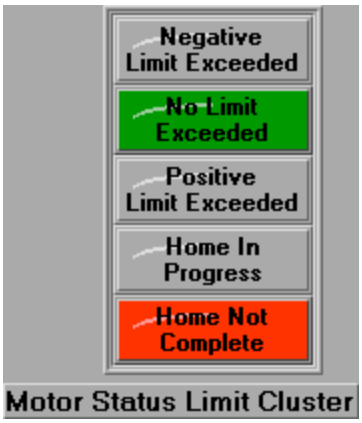
Generally, we have assumed that your PMAC is configured so that Motor N uses Encoder N and Flag N. If this is not the case you must create your own VIs using the pieces provided with PMACPanel or craft your own. In either situation, the architecture and examples presented here will make your life a lot easier.

When you have configured your capture trigger conditions and move direction, velocity, acceleration, etc. set these in PMAC using the "Configure I-Vars" button on the example panel.

Monitoring the Home Position Capture

Homing is a firmware operation that uses the position capture and homing move characteristics just configured. By clicking the Home button in the **PmacMotorLimitControl** cluster, you actually start the movement and monitoring of the encoder status bits. When the movement starts the Home Complete flag for the motor is set to FALSE and the Home In Progress flag is set TRUE. You can see this in the **PmacMotorStatLimit** cluster on the example panel.

- **PmacMotorStatLimit**



At a fundamental level, you can monitor the encoder's operation using the **PmacEncoderStatFlags** cluster and VI. The five indicators on the bottom of the cluster simply reflect the state of their associated inputs. Position Captured indicates that the configured trigger condition, whether used for homing or some other purpose, has occurred. Count Error is used internally by PMAC. Compare-Equal will be covered in the next chapter. A detailed description of these status bits, along with their standard PMAC M-Variable definitions follow.

- **PmacEncoderStatFlags**

Compare-Equal

M116->X:$C000,16,1 ; Compare-equals flag for encoder 1

This compare-equal signal is always copied into the compare-equal flag (M116 here) that is available for PMAC internal use. If you are using this flag internally, make sure that the signal is latched (M111=1), or you will probably miss it. For interrupting the host (edge-triggered), you will probably want the signal transparent.

Position Captured

M103->X:$C003,0,24,S ; Encoder 1 24-bit position capture register

M117->X:$C000,17 ; Encoder 1 position-capture flag

This bit goes TRUE when the trigger condition has gone TRUE; it goes FALSE when the capture register is read (when M103 is used in an expression). As long as the bit is true, the capture function is disabled; you must read the capture register to re-enable the capture function.

Count Error

M118->X:$C000,18,1 ; Count error flag for encoder 1

If an illegal encoder transition (both channels changing on the same SCLK cycle) does get through -- or around, if bypassed -- the delay filter, and to the decoder, a count-error flag (M118 here) is set, noting a loss of position information.

C Channel Status

Quadrature encoders provide an index channel to indicate revolutions of the encoder. This flag is TRUE when the channel is TRUE.

Home Flag

A home switch may be normally open or normally closed; open is high (1 = TRUE), and closed is low (0 = FALSE). The polarity of the edge that causes the home position capture is programmable with Encoder I-Variables 2 and 3 (I902 and I903 for HMFL1).

+/-Limit Flags

When assigned for the dedicated uses, these signals provide important safety and accuracy functions. +LIMn and -LIMn are direction-sensitive over-travel limits, that must be actively held low (sourcing current from the pins to ground) to permit motion in their direction.

The direction sense of +LIMn and -LIMn is the opposite of what many people would consider intuitive. That is, +LIMn should be placed at the negative end of travel, and -LIMn should be placed at the positive end of travel.

Fault Flag

This flag takes a signal from the amplifier so PMAC knows when the amplifier is having problems, and can shut down action. The polarity is programmable with I-variable Ix25 (I125 for motor #1) and the return signal is analog ground (AGND). FAULT1 is pin 49. With the default setup, this signal must actively be pulled low for a fault condition. In this setup, if nothing is wired into this input, PMAC will consider the motor not to be in a fault condition.

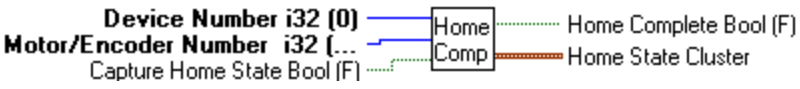
As the homing move proceeds and triggers the physical switch, the encoder will signal this using these status bits. When the configured position capture trigger condition occurs the Position Capture bit will become TRUE.

Home Position Transformations

Monitoring the homing operation is already done in PMAC firmware. When the home move completes, the motor's zero position and its corresponding encoder's zero position will most probably not match. **PmacHomeComplete** monitors the home operation and reports a number of I-Variables and memory registers that both demonstrate what is going on and are used for capture and compare-equal operations in the next chapter.

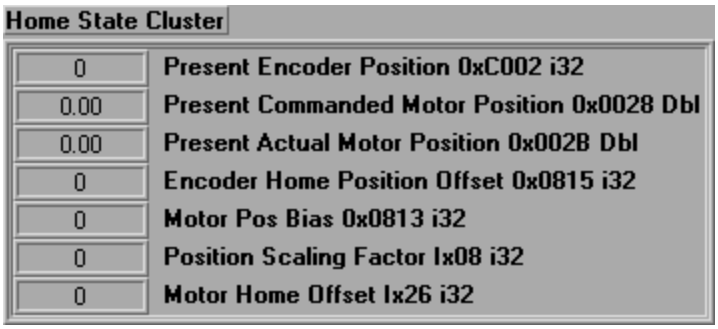
- **PmacHomeComplete** - Create a PmacHomeStateCluster containing I-Variables and memory registers for the specified Motor/Encoder number. The VI monitors the Home In Progress, Home Complete, and Desired Velocity Zero status bits for the motor to determine when to query PMAC for the required data. A query can also be forced if Capture Home State is TRUE.

  This assumes Motor N uses Encoder N.



The contents of the Home State Cluster are discussed in detail here. Again, if you are developing PMACPanel application that uses PMAC's capture or compare-equal capabilities you should understand these quantities.

- **PmacHomeState**



Present Encoder Position 0xC002 i32

The encoder Servo position register is 2 * Encoder counts with the LSB the direction bit. This output value is en actual encoder position referenced to a power-up/reset position of zero.

Present Commanded Motor Position 0x0028 Dbl

This is the motor's present commanded position in units of 1 / (32 * Ix08) encoder counts referenced to the motor's home position.

Present Actual Motor Position 0x002B Dbl

This is the motor's present actual position in units of 1 / (32 * Ix08) encoder counts referenced to the motor's home position.

Encoder Home Position Offset 0x0815 i32

This is the encoder's home offset position in encoder counts. It represents the difference between the encoder's power-up/reset zero position and the position when a home operation completes.

Motor Pos Bias 0x0813 i32

This is the position bias of the motor and represents the coordinate system translation in motor position encoder counts.

Position Scaling Factor Ix08 i32

This parameter controls how the position encoder counter is extended into the full-length register. For most purposes, this is transparent to the user and does not need to be changed from the default.

There are two reasons that the user might want to change this from the default value. First, because it is involved in the "gear ratio" of the position following function -- the ratio is Ix07/Ix08 -- this might be changed (usually raised) to get a more precise ratio.

The second reason to change this parameter (usually lowering it) is to prevent internal saturation at very high gains or count rates (velocity). PMAC's filter will saturate when the velocity in counts/sec multiplied by Ix08 exceeds 256M (268,435,456). This only happens in very rare applications -- the count rate must exceed 2.8 million counts per second before the default value of Ix08 gives a problem.

When changing this parameter, make sure the motor is killed (disabled). Otherwise, a sudden jump will occur, because the internal position registers will have changed. This means that this parameter should not be changed in the middle of an application. If a real-time change in the position-following "gear ratio" is desired, Ix07 should be changed.

In most practical cases, Ix08 should not be set above 1000 because higher values can make the servo filter saturate too easily. If Ix08 is changed, Ix30

should be changed inversely to keep the same servo performance (e.g. if Ix08 is doubled, Ix30 should be halved).

Motor Home Offset Ix26 i32

This is the relative position of the end of the homing cycle to the position at which the home trigger was made. That is, the motor will command a stop at this distance from where it found the home flag(s), and call this commanded location as motor position zero.

This register permits the motor zero position to be different from the home trigger position. It is particularly useful when using over-travel limits for a home flag (offsetting out of the limit before re-enabling the flag as a limit). If large enough (greater than 1/2 times home speed times accel time) it permits a homing move without any reversal of direction.

The units of this parameter are 1/16 of a count, so the value should be 16 times the number of counts between the trigger position and the home zero position.

Example:

If you wish your motor zero position to be 500 counts in the negative direction from the home trigger position, you would set Ix26 to -500 * 16 = -8000.

Encapsulated PLC Programs

In Chapter 6 we introduced VI wrappers that encapsulated motion programs and their operation into a single VI. PMACPanel also encapsulates PLC programs. The discussion of this topic was deferred until here because we now have a good example of their use – homing from a PLC program. The following PLC program, **PmacHomePLC1.pmc,** taken from the *PMAC User Manual* and uses the +LIM flag to establish a home position for motor 1.

```
; PLC Set-up Variables (to be saved)

CLOSE
M133->X:$003D,13,1      ; Desired Velocity Zero bit
M145->Y:$0814,10,1      ; Home complete bit

; PLC program to execute routine

OPEN PLC 10 CLEAR
I123=-10    ; Home speed 10 cts/msec negative

;I125=$C000 ; Use Flags1 for Motor 1 (limits enabled)
I126=32000  ; Home offset of +2000 counts
            ; (enough to take you out of the limit)

I902=3      ; Capture on rising flag and rising index
I903=2      ; Use +LIM1 as flag (negative end switch)
I125=$2C000 ; Disable +/-LIM as limits

CMD"#1HM"   ; Home #1 into limit and offset out of it

WHILE (M145=1)    ; Waits for Home Search to start
ENDWHILE
```

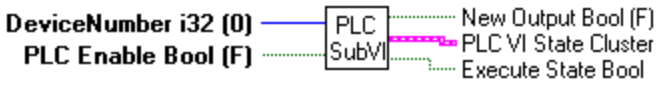
```
WHILE (M133=0)    ; Waits for Home motion to complete
ENDWHILE

I125=$C000  ; Re-enable +/-LIM as limits
DIS PLC10   ; Disables PLC once Home is found
CLOSE ; End of PLC
```

Using **PmacTerminalEdit** you can load this program and click the "Create PLC VI" to create an encapsulated PLC Sub VI for this PLC. This has already been done for this homing PLC, **PmacHomePLC1.vi,** and the other PLC and motion homing programs documented in the *PMAC User Manual*. The raw encapsulated PLC Sub VI is shown here

- **PmacPLCSubVI** - PmacPLCSubVICreate makes a copy of this VI with a new name that matches the name of a PLC program. Because the PLC program has the same name (with a different extension) this VI knows how to open, parse, load, and run a PLC program without intervention or extra inputs. It allows you to interactively monitor and change the PLC program's execution state. Details of its implementation are contained in the manual.

  The VI downloads the associated PLC program when first loaded unless this option is disabled in the diagram and a default for PLC Number are provided for the PLC VI State Cluster.

  The VI queries PMAC for the PLC's execution state every execution. This is done whether the program is executing or not. New Output is TRUE any time PLC Enable is TRUE.



Using this new wrapper VI it is easy to create PLC programs and use them in your PMACPanel applications. The indicator on the example panel displays the Execute State of the properly loaded PLC program every iteration of the VI.

The button "Home PLC 1 Toggle" on the example panel changes the state of the PLC when clicked. For the purposes of this example, if you click the button, the PLC begins executing and the sequence of operations in PLC 10 begin executing thereby configuring and executing the specified home operation.

There is one important point to note about this example. M133 and M145 are defined outside the actual definition of the PLC. When the VI is first executed the entire program buffer, including these statements, is compiled and down loaded to PMAC. If you also happen to set certain I-Variable and memory locations before the OPEN PLC statement these are executed when the program is downloaded. Not every time the PLC is enabled.

We will see a few more examples of encapsulated PLC's in the following chapters.

Chapter 8 - Encoder Capture and Compare Operation

Basics

PMAC provides sophisticated and precise motion capabilities that can be easily accessed from PMACPanel applications. When coupled with National Instruments data acquisition boards, PMAC and PMACPanel can be used to build highly integrated and precise motion based data acquisition systems using GPIB, SCXI, VXI, or DAQ boards. The degree of integration is directly related to your particular system and performance requirements.

Loosely coupled systems with slow event or clock rates that can be handled by LabVIEW can be integrated primarily with PMACPanel VIs and PMAC PLC/motion programs. Tightly coupled systems with fast clock rates or tightly synchronized motion and data acquisition requirements are easily handled using a few terminal blocks and wires to couple the HW systems.

In this Chapter, we will introduce a set of VIs for converting between encoder position and motor position. This is followed by an example extending the position capture capabilities introduced in Chapter 7 demonstrating how you can capture positions in response to NI-DAQ signals, mechanical HW triggers, and clocks and use the captured positions in your application. Finally, we will introduce PMAC's compare-equal capabilities and demonstrate several approaches for generating SW and HW triggers at specific positions while PMAC is in motion. PMAC generated position triggers and clocks can then be used by your NI-DAQ boards to control and synchronize acquisition. In all cases, PMACPanel simplifies the required tasks by allowing you to work in CS units, motor position, or raw encoder units.

In the Chapter 9 we show how to couple standard NI-DAQ boards to PMAC to synchronously trigger data acquisition at specified positions, and even use PMAC's servo clock as your DAQ sampling clock.

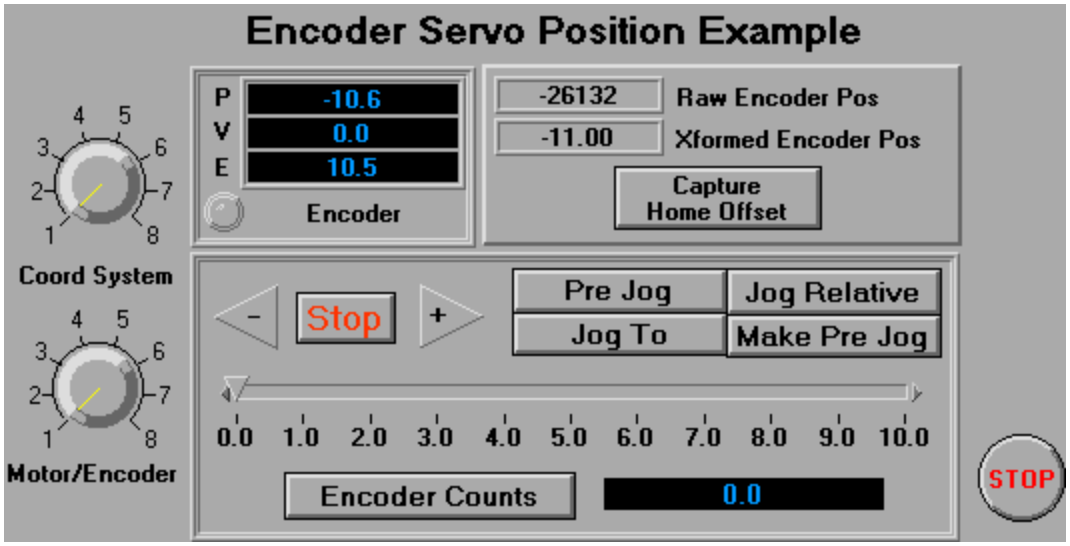
PmacEncoderPositionExamp

This example demonstrates how PMACPanel handles encoder positions. This is important for transforming captured positions into motor position and translating compare positions specified in motor or CS units into encoder position.

The panel, shown below, Motor/Encoder and Coord System knobs, Motor PVE, and a Jog cluster. At the top right of the panel are two indicators that display the

encoder position as the raw encoder position and the encoder position converted into motor position or CS units using VIs in the **PmacEncoder** collection.

Before you run this VI, you should home the motors you are working with. You can do this with the example covered in Chapter 7 or execute a home command from **PmacTerminal.**



Encoder Position Transformations

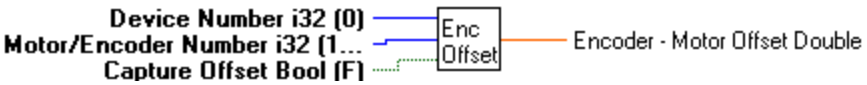
When you execute this example the position indicator in the PVE cluster will display the current motor position. The current encoder position is displayed in the indicator labeled Raw Encoder Pos. Expect these two values to be different as they are in the panel.

The most basic requirement for converting between encoder position and motor position and/or CS units is the determination of the offset between a motor's zero position and the encoder's zero position. The homing operation will generate the necessary data internally to PMAC. The following VI fetches this data and computes an offset to transform between encoder position and motor position.

- **PmacEncoderOffset** - Query PMAC for the encoder to motor offset captured during a home operation for Encoder/Motor Number. This assumes that encoder one is defined for motor 1, etc.

  Encoder-Motor Offset provides a reference for using the encoder position-capture and position-compare registers. These registers are referenced to the encoder zero position, which is the power-up position, not the home (motor zero) position. This value is the difference between the two positions and the home offset Ix26.

  This value should be subtracted from encoder position (usually from position capture) to get motor position, or added to motor position to get encoder position (usually for position compare).



The trick in using this VI is to know when to query PMAC for the Home offset information. You need to do this after you home the motor.

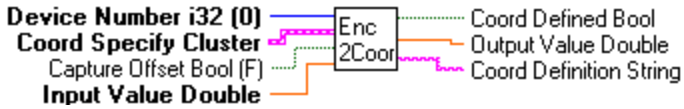
If you click the Capture Home Offset button in the panel, the offsets and biases for the specified motor/encoder number are retrieved. The indicator labeled Xformed Encoder Pos now displays motor position as computed directly from encoder position. There will be a very slight difference between the position in the PVE cluster and the Xformed Encoder Pos value due to the sub-count interpolation used internally by PMAC to compute motor position.

If you click the Encoder Counts button in the Jog cluster, the PVE cluster and the Xformed Encoder Pos indicator will both display motor position in CS units. Again, the accuracy is subject to the interpolation performed by PMAC for its own internal use.

The transformation in both cases is implemented by combining **PmacEncoderOffset** and **PmacCoordMotorToCoord** to build the following VI that converts a raw encoder position (either Capture or Compare) to motor position or CS units. This VI is most often used to convert a captured encoder position into motor position or CS units.

- **PmacEncoderToCoord** - This VI converts Input Value (Servo Position or Capture Position) from absolute encoder position to either CS units or motor position in encoder counts.

  Coord Specify Cluster specifies a motor within a CS and an attempt to convert Input Value from encoder position to CS units. If the motor is not defined in the CS Output Value is motor position in encoder counts. If the motor is defined and Convert is TRUE Coord Defined is TRUE and Output Value is in CS units. Coord Definition is a string specifying Output Value units as "Encoder" or the CS definition of the motor.

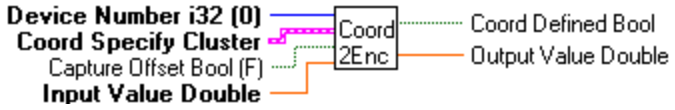


To use this VI you must supply an optional Capture Offset that will fetch and compute the proper offsets. Once the offset is captured and computed, it is maintained by the VI's internal state.

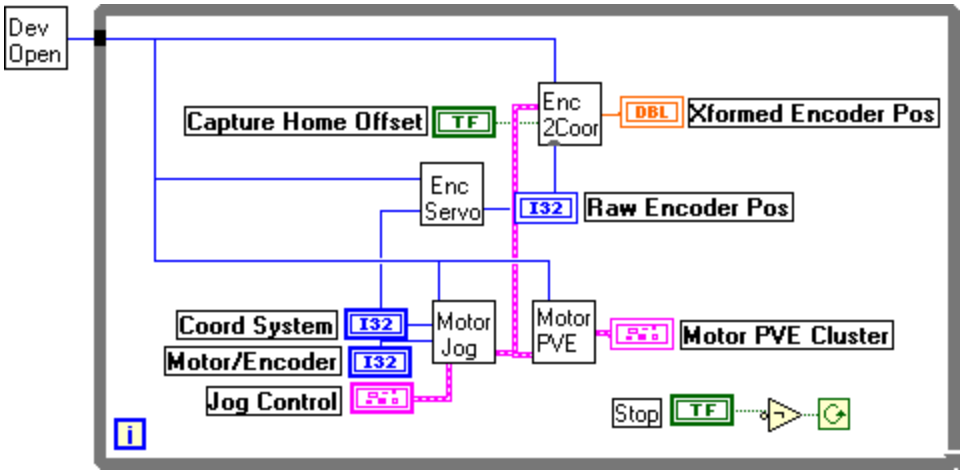
PmacEncoderToCoord has a companion that takes positions specified in CS units or motor position and converts them to encoder position. This VI is most often used to take a motor position in encoder counts or CS units to encoder position for compare-equal operations.

- **PmacEncoderToEncoder** - This VI converts Input Value in either CS units or motor position in encoder counts to an absolute encoder position for compare-equal operations.

  Coord Specify Cluster specifies a motor within a CS and an attempt to convert Input Value from CS units to encoder position. If the motor is not defined in the CS Input Value is assumed to be motor position in encoder counts and Output Value is encoder position. If the motor is defined and Convert is TRUE Coord Defined is TRUE and Output Value is scaled from CS units to encoder position.



The diagram fetches and processes the encoder position two VIs that implement the position transformations just discussed. **PmacEncoderRegServo**, covered later, fetches the encoder position, not capture positions, directly from the encoder, and displays it on the panel. **PmacEncoderToCoord** uses the transformations discussed above to compute Xformed Encoder Pos directly from this encoder position.



Position-Capture for Non-Homing Purposes

Chapter 7 introduced the encoder architecture and its use for homing operations. Homing is a firmware command and therefore does not require you to monitor the capture flags, access the capture register, or do anything with the value. To use the position capture function for operations other than homing in your own program you need to

- Configure the capture condition
- Monitor the capture flag
- Process the capture register
- You can do this using a PLC or using PMACPanel directly.

PLC Capture Flag Processing

If you use a PLC to handle the capture operation you need to monitor the position-captured flag bit -- bit 17 of the encoder control/status register using

```
M117->X:$C000,17,1
```

and the captured position using the M-Variable

```
M103->X:$C003,0,24,S
```

This status bit turns TRUE when the trigger condition turns TRUE. It returns to a non-triggered FALSE state when the capture register (M103) is read. As long as the status bit is TRUE, the capture function is disabled; you must read the capture register to re-enable the capture function. The example program **MOVTRIG.PMC** in the *PMAC User Manual* shows how this capability can be used for precision registration.

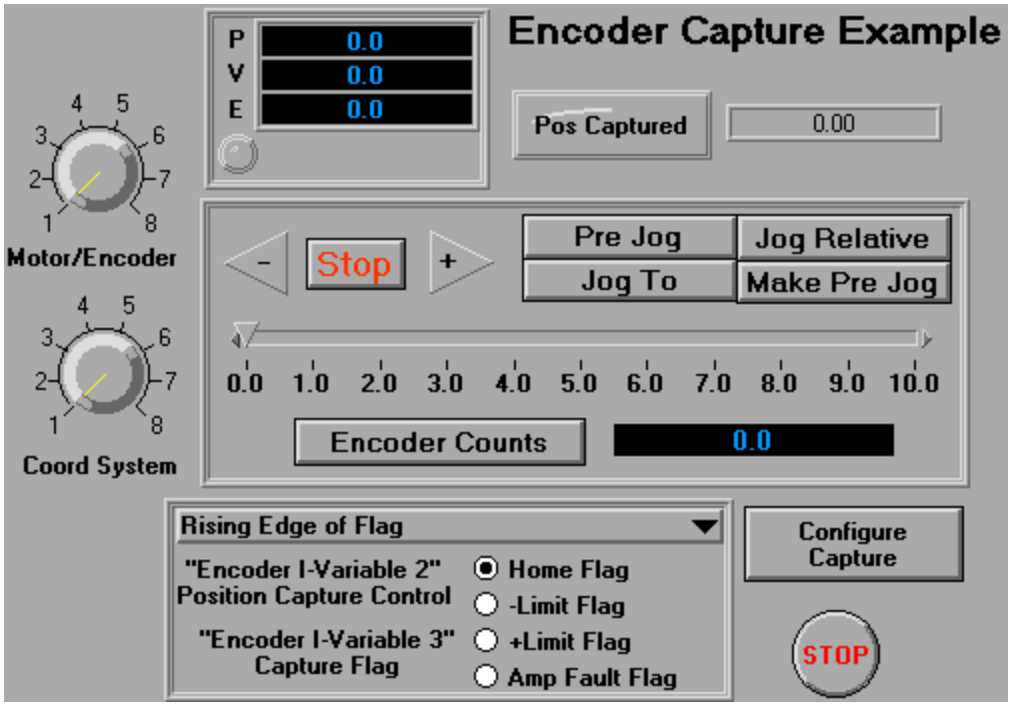
In the example that follows, we will show precisely how PMACPanel can be used to add capture capability to your application. We will not cover an example of position capture handling using a PLC. This is only required if multiple captures occur faster than PMACPanel can service them or your motion program is using them directly.

PmacEncoderCaptureExamp

This example demonstrates how PMACPanel handles encoder capture operations. This is important when you want to determine the position of a motor when a trigger occurs in your system.

The panel, shown below, has Motor/Encoder and Coord System knobs, a Motor PVE indicator, and a Jog cluster. At the top right of the panel is an LED that turns Green when an externally triggered capture trigger occurs and a position indicator whose value is the position captured when the trigger occurs.

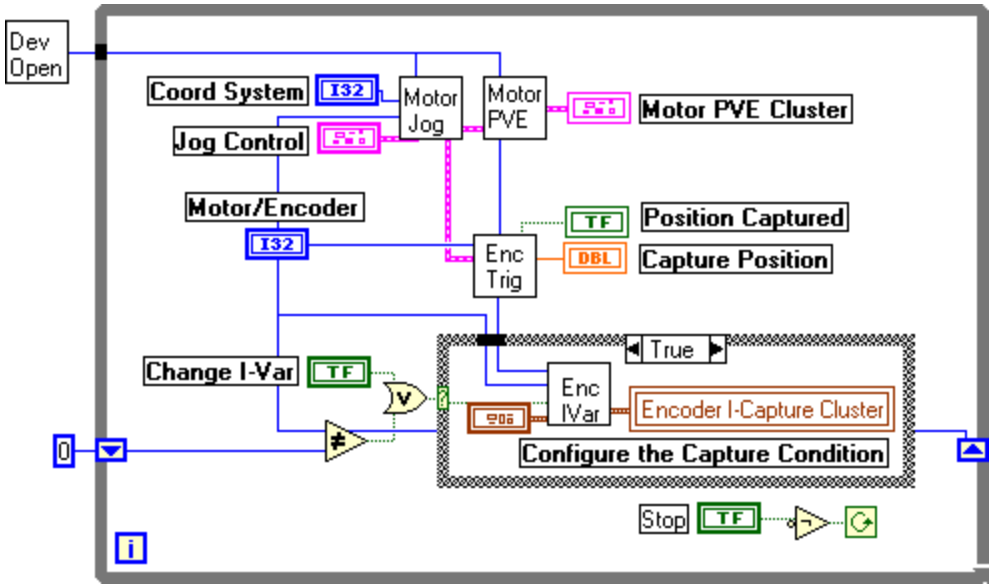
Before you run this VI, you should home the motors you are working with. You can do this with the example covered in Chapter 7 or execute a home command from **PmacTerminal.**



As with homing position capture the trigger condition must be configured prior to use. This is done using the already introduced **PmacEncoderIVarCapture** cluster. The **PmacHomeIVar** cluster is not required because the motor has been homed and the moves we will be executing are not homing moves.

After selecting the capture trigger condition, the Configure Capture button sets the configuration in PMAC. Once this is done, the encoder is armed and waiting for the specified capture trigger.

The simplicity of the diagram demonstrates how the application is organized. At the top are the VIs to handle the jog and PVE clusters. At the bottom is the logic to handle the configuration of the capture condition. The encoder capture trigger condition is configured whenever the motor number changes or you click the Configure Capture button.



Once the capture condition is configured, **PmacEncoderTrigger** is used to monitor the encoder flags. When a trigger occurs the VI reads the capture register and transforms the captured position into motor position or CS units.

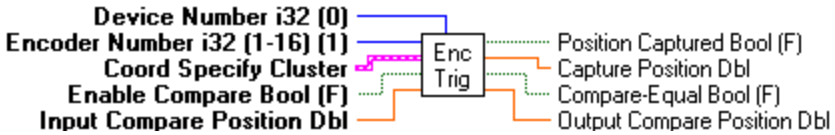
The VI does a lot of bookkeeping to make your job easier. To use it for capture operations leave the Enable Compare and Input Compare Position terminals unwired. In the next section on Compare operations, we will see how this VI also handles compare triggers.

- **PmacEncoderTrigger** - This VI maintains Encoder Number's compare-equal and capture operations and monitors the encoder's status register. Home offsets are removed or added during the processing of compare-equal and capture register data. Limitations associated with 24 bit rollover are not handled by this VI.

  When Enable Compare is TRUE Encoder Number's compare-equal function is reset and the compare-equal register is set using Input Compare Position. This value is interpreted as being in CS units if Covert is TRUE and Motor Number is defined in Coord Number. Otherwise this value is interpreted as being motor position in encoder counts. Output Compare Position is a persistent copy of Input Compare Position when Enable Compare was TRUE. The occurrence of the compare-equal condition is indicated by

Compare-Equal Bool being TRUE. This does not reset the latched condition.

When Encoder Number captures a position, Position Captured is TRUE, and the encoder's capture register is queried and converted into Capture Position in motor position encoder counts or Coord Number CS units. If Motor Number is not defined in Coord Number or Convert Bool is FALSE the Capture Position is mo tor position in encoder counts . If Convert Bool is TRUE and Motor Number is defined in Coord Number the value is in CS units.



External Triggers for Position Capture

Using the HOME, +/-LIM, or FAULT flags for other than their obvious purpose is very common on PMAC. It requires you to build a simple interface circuit to disconnect the physical limit or flag switches and connect the trigger signal of your choice. For example, your system may have a proximity switch with TTL output to define HOME. When you want to use the HMFL input for your own position capture operations, a TTL MUX or other form of digital selector can be used to connect the trigger signal you desire. Realize that you do not home the motor using this new trigger signal. You will be using it for position capture for registration or some other purpose.

In the system used to develop and demonstrate PMACPanel's capture capabilities a TTL signal generator was used to drive a reed relay at 5 Hz and trigger the home flag. Using this configuration the Pos Captured LED flashes on and off dutifully signaling the capture of the position. When the capture condition occurs PMACPanel reads the capture register and converts it into motor position or CS units depending on whether the Encoder Counts button in the Jog cluster is activated. When the motor is jogging the numeric updates with every tick of the HOME trigger signal.

PMAC Position Compare Operation

PMAC's encoder position-compare function is essentially the opposite of the position-capture function. Instead of capturing the encoder position when an external signal changes, compare operations generate a signal when the encoder reaches a specified position. In fact, the encoder register into which the captured position is written is used to store the position for compare operation. Using this capability, you can configure trigger events that occur at specified encoder positions in your system. Because the triggering is implemented in hardware, it is very fast and accurate. In Chapter 9, we show how to use external TTL level signals to trigger data acquisition on NI-DAQ boards.

Compare operations require three steps

1. Enable and configure the operation using the encoder control register

2. Load the compare position into the encoder register

3. Monitor the compare-equal flag in the encoder status register and repeat these steps as required

These steps may be performed in a PLC or a PMACPanel program.

Required M-Variables

To utilize this feature from a PLC, you must access the encoder control/status register and the position compare-equal register. For Encoder 1, the standard M-Variable declarations are

```
M103->X:$C003,0,24,S    ;24-bit pos compare register
M111->X:$C000,11,1      ; Compare flag latch control
M112->X:$C000,12,1      ; Compare output-enable bit
M113->X:$C000,13,1      ; Compare output invert bit
M116->X:$C000,16,1      ; Compare-equals flag
```

Similar sets of registers and M-Variables are defined for the other encoder registers.

Pre-loading the Compare Position

To pre-load a compare position, assign an encoder position value to M103, such as `M103=1250`. This value must be between -8,388,608 and +8,388,607. You cannot read this value back; reading from the same address gives you the position-capture register. The command can be given from a PMAC motion program, a PMAC PLC program, from the host, or using the PMACPanel VIs introduced in the following example. This is the *encoder* position; if you want to reference it to motor zero position, you must know the homing offset. This translation is handled transparently by the PMACPanel ICVs in the **PmacEncoder** collection.

Encoder Control Bits

Three control bits configure the format of operation of the compare feature. These are SW status bits and, if enabled, external HW signals available on various PMAC cables. The flag-latch control bit (M111) controls whether the compare-equal signal is

- **Transparent** -- TRUE only when the positions are actually equal
- **Latched** -- TRUE until actively reset by a handler

The signal is transparent if this control bit is zero, and latched if the control bit is one. To clear a latched flag, take the control bit to zero then back to one.

The compare-equal signal is always copied into the compare-equal flag (M116). If you are monitoring this flag from a PLC or PMACPanel application, make sure that the signal is latched (M111=1), or you will probably miss it. To interrupt the host (edge-triggered), you will probably want the signal transparent. PMACPanel doesn't currently support an interrupt driven interface. Look for this in a future release.

The output-enable bit (M112) determines whether the compare-equal flag will be output on the PMAC EQU line (1 enables). This must be set if you want to use the signal to interrupt the host or to trigger an external event. The output-

invert bit (M113) determines whether the EQU output is high-true or low-true (1 inverts -- low-true). For host-interrupt purposes, this must be configured high-true.

Triggering External Action

To trigger external actions from a PMAC-PC, you should put a connector on the E-points (E53-E65) that normally jumper these signals to the interrupt controller. An IDC 26-pin connector works nicely. These signals must be buffered; the TTL drivers for these outputs on PMAC-PC are very weak. You can obtain an application note on techniques for accessing these signals by contacting Delta Tau technical support.

On the PMAC-Lite, PMAC-VME and PMAC-STD, a JEQU connector provides direct access to the Compare-Equals signals. The outputs are open-collector (sinking) outputs, rated to 24V and 100 mA. The user may replace the existing driver IC with a sourcing driver IC (UDN2981A).

To use these HW signals, and several others, you must refer to the *PMAC User Manual*. We will cover their use as far as NI-DAQ boards are concerned in Chapter 9.

PLC Compare Handling

The PLC programs **PmacPosCompSetup.pmc** and **PmacPosCompGen.pmc** located in **\PmacEncoder** demonstrate the use of a PLC to generate a very rapid series of "equals" pulses at specified position intervals. PLC's are an excellent way to handle compare operations that require fast servicing. You will find these documented in the *PMAC User Manual*, PMAC application notes, and in the Introduction to PMAC tutorial notes. **PmacPosCompSetup** configures the capability by fetching the current encoder position, adding the interval, and initializing the encoder registers. This PLC is executed once to configure the operation. After configuring the operation it starts **PmacPosCompGen.** This PLC monitors the encoder's compare-equal flag. When the specified position is reached, it clears the flag, loads the next compare position, and calculates the next position to be used.

PmacPosCompSetup.pmc

```
close

; Define encoder registers

m101->x:$c001,0,24,s ; Actual position
m103->x:$c003,0,24,s ; Compare register
m105->x:$07f0,0,24,s ; Scratch register for rollover

; Define encoder compare-equal register control bits

m111->x:$c000,11,1      ; Compare equal latch/control
m112->x:$c000,12,1      ; Compare equal output enable
m113->x:$c000,13,1      ; Compare equal output invert
m116->x:$c000,16,1      ; Compare equal flag

p101 = 50               ; Count intecrement

; Configure the compare pulse

open plc 18
clear
```

```
; -- Setup compare-equal

m105 = m101+p101  ; Save Increment + actual position
m103 = m105       ; Copy next pos into compare reg
m105 = m105 + p101    ; Update next compare position
m113 = 0              ; No invert on output bit

enable plc 19
disable plc 18

close
```

PmacPosCompGen.pmc

```
close

; - Service routine to service encoder register

open plc 19
clear

if (m116 = 1)
  m103 = m105     ; Update next compare position
  m105 = m105 + p101
  m111 = 0        ; Reset control bit
  m111 = 1
endif
close
```

The PLCs can be downloaded and executed using **PmacTerminalEdit** and **PmacTerminal** or, as we will show in a moment, encapsulated with a wrapper VI and controlled from an application panel.

PmacEncoderCompareExamp

This example demonstrates three methods for using PMACPanel to handle encoder compare operation. These are extremely useful for synchronizing data acquisition operations with complex motion. The three methods are

- Using encapsulated versions of the **PmacPosCompSetup** and **PmacPosComGen** PLC's to generate position interval clocks.
- Directly setting an encoder compare position from your application for a one-time position-compare trigger.
- By servicing the control, status, and position registers directly from PMACPanel. This is a poor-man's approach to using the PLCs.

The panel, shown below, has Motor/Encoder and Coord System knobs, a Motor PVE indicator, and a Jog control cluster. At the top left of the panel is an LED that flashes green when a compare-equal condition occurs. Below this is a cluster of three buttons that allow you to configure the encoder control bits. When these are properly set clicking the Configure Compare button sets the encoder bits.

The remaining controls are divided into three groups. One for each method demonstrated in the example.

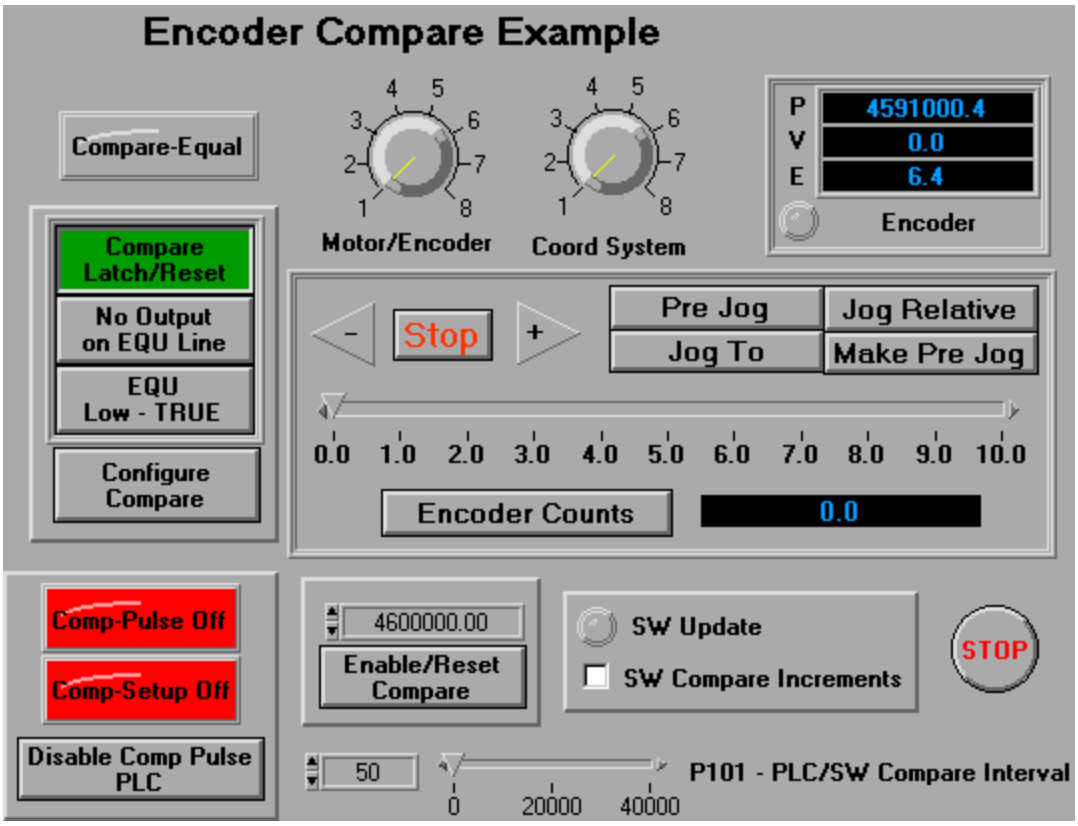
- On the bottom left are LEDs indicating the Execution State of the two PLCs used to service the encoder and a button to disable the PLC handler.

- To the right is a numeric control used to specify a compare position and a button to configure the encoder for one-time compare operation.

- To the right of this is a checkbox that enables encoder servicing directly from PMACPanel – not the PLC. The LED indicates when a new compare position is being loaded into the encoder after a compare-equal trigger occurs.

On the very bottom is a slider that specifies the interval between generated triggers. This interval is used by the PLCs and by the SW interval generation.

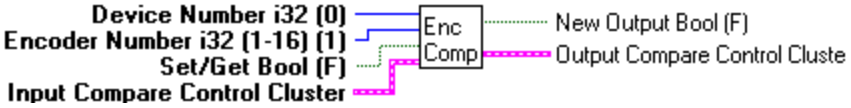
Before you run this VI, you should home the motors you are working with. You can do this with the example covered in Chapter 7 or execute a home command from **PmacTerminal.**

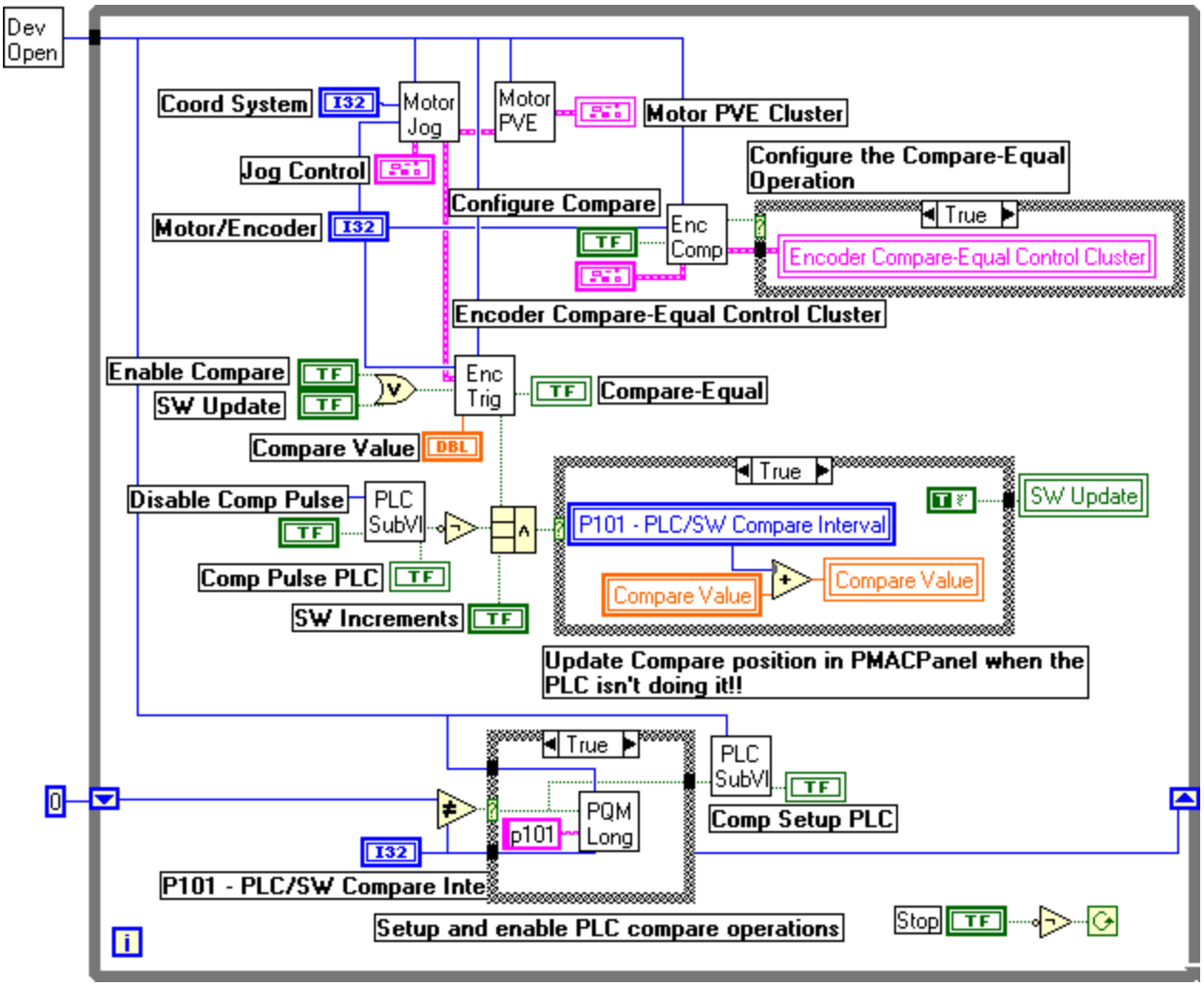


Detailed descriptions for operating the three encoder-handling methods are covered later. At the top of the diagram are VIs to handle the Jog control and PVE indicator. Below this, **PmacEncoderCompareConfig** configures the encoder's compare control bits when the Configure Compare button on the panel is clicked. Configuration can be done by the PLC.

- **PmacEncoderCompareConfig** - Follow PMACPanel's standard I-Variable VI architecture. When Set/Get is TRUE the Input Compare Control bits for

the specified Encoder Number are set. Otherwise they are fetched from PMAC and provided by Output Compare Control Cluster with New Output TRUE. Set/Get is not required and defaults to a Get operation.



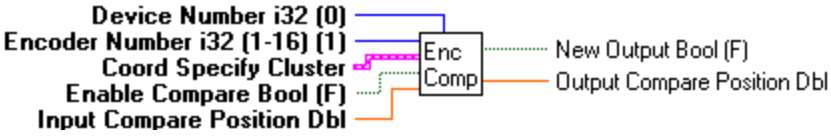


At the bottom of the diagram is logic to service the panel's P101 slider when it changes. Changes in the value update P101 and enable the encapsulated PLC **PmacPosCompSetup** covered earlier. Above this is the wrapper VI for **PmacPosCompGen**. The panel button Disable Comp Pulse can be used to turn the PLC on and off as desired. When this PLC is not executing, a compare trigger occurs, and SW increments is TRUE the case to the right executes and computes the next Compare Value. The final piece of the example is **PmacEncoderTrigger** also covered earlier. In this example the Enable Compare input is TRUE whenever Enable Compare or SW Update are TRUE. This updates the encoder registers thereby arming the compare operation.

We've already noted that **PmacEncoderTrigger** handles the configuration of compare operations and monitoring of capture and compare flags. The implementation of the VI is complex so it is not covered here. However, one of its pieces may be of use in your application. The following VI is used by **PmacEncoderTrigger** to enable and configure compare operations.

- **PmacEncoderCompare** - This VI reset Encoder Number's compare-equal function and set the position register using Input Compare Position when Enable Compare is TRUE. This value is interpreted as being in CS units if Covert is TRUE and Motor Number is defined in Coord Number. Otherwise this value is interpreted as being motor position in encoder counts. Home offsets are removed prior to setting the encoders actual register value.

  Limitations associated with 24 bit rollover are not handled by this VI Output Compare Position is a persistent copy of Input Compare Position when Enable Compare was TRUE.



One last word on the use of **PmacEncoderTrigger** is needed. If your application uses PLCs to handle the capture or compare triggers you should not service them with your PMACPanel application. The chance of getting into trouble having two sets of handlers for a capture or compare operation is pretty large. This does not prevent you configuring the operations using PMACPanel and servicing them with a PLC. You should simply be aware of who is responsible for handling the encoder.

Method 1 - PLC Operation

The P101 slider specifies the interval the PLCs will use to generate compare-equal triggers. Changing this value sets P101 in PMAC and enables **PmacPosCompSetup** discussed earlier. This PLC captures the current encoder position, adds the interval to the position, sets the compare-equal register, and resets the compare-equal control bits. The PLC enables the **PmacPosCompGen** PLC and disables itself. Thus, when you change P101 the Comp-Setup LED briefly turns Green to indicate that the setup PLC is executing. When it enables **PmacPosCompGen** the Comp-Pulse LED turns Green to indicate that it is active, then the Comp-Setup LED turns Red to indicate that it has disabled itself. This is all done using the encapsulated PLC Sub VIs.

If you now jog the motor, the **PmacPosCompGen** PLC will generate compare-equal pulses every P101 encoder counts. It does this by monitoring the encoder Compare-Equal flag for the TRUE condition, setting the next position, adding the increment for the next position, and resetting the encoder's compare-equal control bits. Because the PLC handles the flags PMACPanel never (almost never) sees the compare-equal condition because the PLC services the trigger so quickly. Even if **PmacEncoderTrigger** sees the trigger condition, it does not service it.

You can bring the trigger to the external world by clicking the Output on EQU Line button and then clicking Configure Compare. You should do this before starting the Jog or enabling **PmacPosCompSetup** so that you don't interfere with **PmacPosCompGen's** handling of the encoder flags. If you configure the

external EQU signal you can connect an oscilloscope to the appropriate pins on JEQU or the E-Point jumpers documented in the PMAC Users Manual and see the generation of the interval pulses. In Chapter 9, we will demonstrate how to use these pulses to synchronize your DAQ systems with your system's motion. As you increase P101 the time between the pulses increases. When you stop the jog, the pulse interval increases as the motor slows and eventually ceases because the motor stops. When you begin a jog, the pulse interval decreases until the motor reaches a steady state velocity.

Method 2 - One-Shot Operation

If you disable the **PmacPosCompGen** PLC by clicking the Disable Comp Pulse PLC button, the corresponding indicator turns RED. You can now manually configure compare-equal operations by entering a position in the numeric control above the Enable/Reset Compare button. The value you enter, in motor position or CS units as specified by the Encoder Counts button in the Jog cluster, is used to configure a one-time compare-trigger.

Select the conditions for the operation using the buttons in the configuration cluster. You will most probably want the operation latched. You can send the trigger to the external world using Output on EQU Line and configure whether the condition is TRUE High or TRUE-Low. When you've done this click Configure Compare then Enable/Reset Compare. This will configure the flags and the compare-equal value. When you jog the motor, the Compare-Equal indicator will turn Green when the condition occurs. You cannot read the compare-equal register so you need to keep track of the last value you set. Fortunately, PmacEncoderTrigger does this for you.

Method 3 - PMACPanel Interval Generation

You can perform the same interval generation done by the PLC's using PMACPanel. This works only when the interval rate is long relative to LabVIEW's service rate. If you miss an interval and the motor is already beyond the next interval position, the compare condition never occurs and you stop generating pulses.

If you disable the **PmacPosCompGen** PLC and check the SW Compare Interval box the trigger is handled by the 3-input AND case. Be careful that you don't enable the PLC's by changing P101 before you get the SW version of the interval generation running.

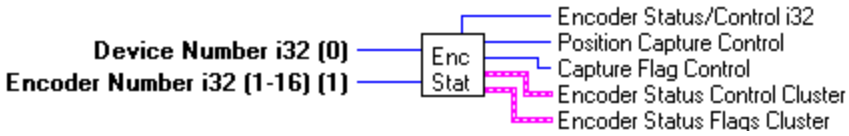
PmacEncoder Registers

Incorporating compare and capture capabilities into your own applications is facilitated by the **PmacEncoder** collection of ICVs. These fall into three categories. Basic encoder register access and control, conversion of encoder positions into motor position or CS units and back, and ICVs to facilitate your application. These are all documented here.

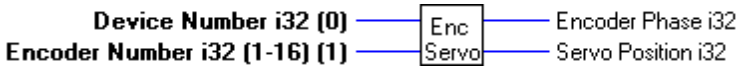
Encoder Register Access

VIs to access the encoder registers are provided but are generally not used in your applications. They are, however, very useful when beginning to work with the encoders. The most important of these are

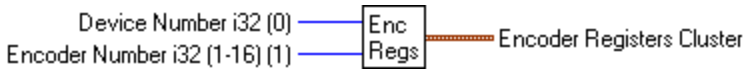
- **PmacEncoderRegStat** - Fetch the encoder control/status word for Encoder Number and parse it into its pieces. Encoder Status/Control i32 is the integer representation of the register. Position Capture Control can be used with PmacEncoderCaptureControl. Capture Flag Control can be used with PmacEncoderCaptureFlag. Encoder Status Control Cluster can used with PmacEncoderStatControl. Encoder Status Flag Cluster can be used with PmacEncoderStatFlags.



- **PmacEncoderRegServo** - Query PMAC for the two position registers containing commutation phase and servo position. Servo Position is actual encoder position in counts referenced to a power-up/reset position of zero. Encoder Phase is used internally for commutation.



- **PmacEncoderRegisters** - Query PMAC for all registers for Encoder Number. Assemble the values into a PmacEncoderRegisters Cluster.



The remaining members of this collection will not generally be used in your application but are provided for completeness. These are

- **PmacEncoderRegTime**
- **PmacEncoderRegDAC**
- **PmacEncoderRegCapture**
- **PmacEncoderRegADC**

Chapter 9 - PMAC and NI-DAQ Interfacing

Basics

In Chapters 7 and 8 we introduced PMAC's position capture and compare capabilities. PMACPanel provides a number of ICVs to configure, monitor, and operate these capabilities. When coupled with National Instrument's data acquisition boards, PMAC and PMACPanel can be used to build highly integrated and precise motion based data acquisition systems using GPIB, SCXI, VXI, or DAQ boards.

In this chapter we will demonstrate how to couple standard NI-DAQ boards to PMAC to synchronously trigger data acquisition at specified positions, and even use PMAC's servo clock as your DAQ sampling clock. The examples presented here in no way limit the wide array of possibilities or approaches you can use to trigger, synchronize, and organize your motion based DAQ applications. Your experience and requirements will define the approaches that best meet your needs.

The examples of PMAC-DAQ interfacing assume that you have a basic understanding of LabVIEW's data acquisition capabilities and the acquisition boards you will be using. Similarly, you must have some understanding of PMAC's external connectors and their configuration. You will be making connections between these boards. If you are not certain of your abilities, precisely which signals you need, where to locate them, or what to connect them to do not attempt to connect them. You can easily damage the boards, the host computers, and many other items. Make certain you have thoroughly studied the information presented in this chapter and the HW manuals provided for your model of PMAC and your DAQ boards. Contact Delta Tau or National Instruments Technical Support if you have any questions prior to proceeding.

External PMAC Signals

PMAC provides a number of HW interfaces that can easily be used to synchronize PMAC with most NI-DAQ boards and systems. Depending on your needs PMAC also supports a number of I/O accessories. PMACPanel doesn't support all of these with ICVs. It is way beyond the scope of this manual to detail all possible approaches to accessing the interfaces on these boards. You can easily create your own ICVs for these accessories using **PmacAcc** and **PmacMemory** ICVs.

We will consider three primary PMAC signals here. Position Capture was discussed in Chapter 7 and will not be repeated here. When interfacing these signals to DAQ boards we will demonstrate how to use these signals to trigger acquisitions and provide the sample scan clock. Before actually using these HW signals you must consult the *PMAC User Manual* and the HW manual for your particular PMAC model. The signals are:

- EQU signals
- Servo Clock
- General Purpose Machine I/O
- Position Capture Flags

We will not consider PMAC's encoder clock or ADC clocks. If you wish to use these consult the *PMAC User Manual*, the HW manual for your particular PMAC model, or contact Delta-Tau technical support.

The following sections are reproduced from various portions of the *PMAC User Manual* and describe the signals, how to access them, and potential limitations in their use.

Compare-Equals Outputs (JEQU)

The compare-equals (EQU) outputs provide a signal edge when an encoder position reaches a pre-loaded value.

PMAC-PC

PMAC-PC doesn't have a dedicated connector for the EQU outputs. Instead, the signals may be accessed using a 26-pin IDC connector over E-point pairs E53-E65. The outputs are TTL-level with very low drive capability; they must be buffered externally before they can drive any real devices. ACC-27, normally used as an I/O buffer for the thumbwheel multiplexer port, can be used to drive several of these EQU lines. The 26-pin cable provided with the ACC-27 fits over the 13 jumper pairs E53-E65. Contact Delta Tau technical support for details.

PMAC-VME

On PMAC-VME, these signals are brought out on connector J7 (JEQU), referenced to digital ground (GND). As shipped from the factory, they are open-collector (sinking) outputs, with a ULN2803A driver IC, rated to 24V and 100mA each. They may be changed to open-emitter (sourcing) drivers by replacing this chip in U28 with a UDN2981A driver IC and changing jumpers E93 and E94.

PMAC-Lite

On PMAC-Lite, these signals are brought out on connector J8 (JEQU), optically isolated from the digital circuitry, referenced either to analog ground (AGND) or an external flag supply ground. As shipped from the factory, they are open-collector (sinking) outputs, with a ULN2803A driver IC, rated to 24V and 100mA each. They may be changed to open-emitter (sourcing) drivers by replacing this chip in U54 with a UDN2981A driver IC and changing jumpers E101 and E102.

PMAC-STD

On PMAC-STD, these signals are brought out on connector J6 (JEQU) on each of the piggyback boards. They are open-collector (sinking) outputs with internal 1-kΩ pull-up resistors, rated to 5V.

On PMAC-STD1.5, these signals are brought out on connector J8 (JEQU), optically isolated from the digital circuitry, referenced either to analog ground (AGND) or an external flag supply ground. As shipped from the factory, they are open-collector (sinking) outputs, with a ULN2803A driver IC, rated to 24V and 100mA each. They may be changed to open-emitter (sourcing) drivers by replacing this chip in U54 with a UDN2981A driver IC and changing jumpers E101 and E102.

Servo Clock (JRS232)

PMAC's servo clock defines the rate at which servo loops are updated and background computations are performed. Using this clock for other timing is an excellent way to synchronize PMAC's movement with externally gathered data.

PMAC allows multiple cards to share a common servo clock over spare lines on the serial connector J4. The servo clock on J4 (JRS232) is located on pin 8 and is referenced to ground on pin 9. If multiple PMACs are being used the clock signals can be shared simply by tying identical pins on the PMACs together. Accessory 3D or 3L cables with extra PMAC connectors (one Accessory 3E for each extra PMAC) can be used to share the clock signals in either bus or serial communications applications (and of course, for actual serial communications). In a standalone or bus-communications application, there is no need for a host drop on the cable. As is the case for the communications lines, you cannot tie the clock lines from the RS-422 port of a PMAC-PC to the RS-232 port of a PMAC-Lite. With the RS-422 option on the PMAC-Lite (Opt. 9L), connection to a PMAC-PC is possible, but the connector pinouts are different.

If serial communication is not being used, but the serial data lines are connected with the clock signals, it may be desirable to deactivate the serial port to prevent noise on the lines from creating input command characters to PMAC. On PMAC-PC, PMAC-Lite, and PMAC-VME, this is done by setting jumpers E44-E47 ON; on PMAC-STD, by making DIP switches SW1-5 to SW1-8 all OFF.

Be aware of the fact that J4 has +5 VDC on pin 10.

General Purpose Digital Inputs and Outputs

The PMAC JOPTO connector (J5 on PMAC-PC, -Lite, and -VME) provides eight general-purpose digital inputs and eight general-purpose digital outputs. Each input and each output has its own corresponding ground pin in the opposite row. The 34-pin connector was designed for easy interface to OPTO-22 or equivalent optically isolated I/O modules. Delta Tau's Accessory 21F is a six-foot cable for this purpose.

The PMAC-STD has a different form of this connector from the other versions of PMAC. Its JOPT connector (J4 on the baseboard) has 24 I/O, individually selectable in software as inputs or outputs. The rest of this discussion does not pertain to the PMAC-STD port, unless specifically mentioned. Refer to the PMAC-STD Hardware Reference for details on its JOPT port.

Standard Sinking Outputs

CAUTION Having Jumpers E1 and E2 set wrong can damage the IC.

CAUTION Do not connect these outputs directly to the supply voltage, or damage to the PMAC will result from excessive current draw.

PMAC is shipped standard with a ULN2803A sinking (open-collector) output IC for the eight outputs. These outputs can sink up to 100 mA, but must have a pull-up resistor to go high.

The user can provide a high-side voltage (+5 to +24V) into Pin 33 of the JOPTO connector, and allow this to pull up the outputs by connecting pins 1 and 2 of Jumper E1. Jumper E2 must also connect pins 1 and 2 for a ULN2803A sinking output.

Option for Sourcing Outputs

CAUTION Having Jumpers E1 and E2 set wrong can damage the IC.

It is possible for these outputs to be sourcing drivers by substituting a UDN2981A IC for the ULN2803A. This IC (U3 on the PMAC-PC, U26 on the PMAC-Lite, U33 on the PMAC-VME) is socketed, and so may easily be replaced. For this driver, pull-down resistors should be used. With a UDN2981A driver IC, Jumper E1 must connect pins 2 and 3, and Jumper E2 must connect pins 2 and 3.

Input Source/Sink Control

Jumper E7 controls the configuration of the eight inputs. If it connects pins 1 and 2 (the default setting), the inputs are biased to +5V for the "OFF" state, and they must be pulled low for the "ON" state. If E7 connects pins 2 and 3, the inputs are biased to ground for the "OFF" state, and must be pulled high for the "ON" state. In either case, a high voltage is interpreted as a '0' by the PMAC software, and a low voltage is interpreted as a '1'.

Memory Mapped Access to I/O

These inputs and outputs are typically accessed with M-variables. In the suggested set of M-variable definitions, variables M1 through M8 are used to access outputs 1 through 8, respectively, and M11 through M18 to access inputs 1 through 8, respectively. This port maps into PMAC memory space at Y address $FFC2. You can also find a collection of VIs to access these in **PmacAcc**.

Synchronous M-Variables

In a motion program, when PMAC is blending or splining moves together, it must be calculating in the program ahead of the actual point of movement. This is necessary in order to be able to blend moves together, and to be able to do reasonable velocity and acceleration limiting. Depending on the mode of movement, calculations can be one, two, or three moves ahead of the actual movement.

Why Synchronous M-Variables are Needed

When assigning values to variables is part of the calculation, the variables will get their new values ahead of their place in the program when looking at actual move execution. For P and Q-variables, this is generally not a problem, because they exist only to aid further motion *calculations.* However, for M-variables, particularly outputs, this can be a problem, because with a normal variable value

assignment statement, the action will take place sooner than is expected, looking at the statement's place in the program.

For example, in the program segment

```
X10   ; Move X-axis to 10
M1=1  ; Turn on Output 1
X20   ; Move X-axis to 20
```

you might expect that Output 1 would be turned on at the time the X-axis reached position 10. Because PMAC is calculating ahead, at the *beginning* of the move to X10, it will have already calculated through the program to the next move, working through all program statements in between, including **`M1=1`**, which turns on the output. Therefore, using this technique, the output will be turned on sooner than desired.

How They Work

With synchronous assignment, the actual assignment is performed where the blending to the new move *begins*, which is generally ahead of the programmed point. In LINEAR and CIRCLE mode moves, this blending occurs V*TA/2 distance ahead of the specified intermediate point, where V is the commanded velocity of the axis, and TA is the acceleration (blending) time.

Synchronous M-variables after the last move or **DWELL** in the program do not execute when the program ends or temporarily stops. Use a **DWELL** as the last statement of the program to execute these statements.

Synchronous M-variable assignment statements were implemented as a solution to this problem. When one of these statements is encountered in the program, it is not executed immediately; rather, the action is put on a stack for execution at the start of the actual execution of the next move in the program. This makes the output action properly synchronous with the motion action.

In the modified program segment

```
X10   ; Move X-axis to 10
M1==1 ; Turn on Output 1 synchronously
X20   ; Move X-axis to 20
```

the statement **M1==1** (the double-equals indicates synchronous assignment) is encountered at the *beginning* of the move to X10, but the action is not actually performed until the start of blending into the next move (X20).

Also, notice that the assignment is synchronous with the *commanded* position, not necessarily the *actual* position. It is the responsibility of the servo loop to make the commanded and actual positions match closely

In applications where PMAC is executing segmented moves (I13>0), the synchronous M-variables are executed at the start of the first I13 spline segment after the start of blending into the programmed move.

Syntax

There are four forms of synchronous M-variable assignment statements:

```
M{constant}=={expr}     ;Straight equals assignment
M{constant}&={expr}     ; AND-equals assignment
M{constant}|={expr}     ; OR-equals assignment
M{constant}^={expr}     ; XOR-equals assignment
```

In all of these forms, the expression on the right side of the statement is evaluated when the line is encountered in the program, ahead of the execution of the move. The value of the expression, the variable number, and the operator are placed on a stack for execution at the proper time.

Position Capture FLAGs

Interfacing to the FLAG inputs required for position capture was covered in detail in Chapter 7 and 8. Because these inputs are so closely associates with HW limit switches you should refer to your PMAC HW *and PMAC User Manual* as well as any manuals supplied by your system integrator. Many PMAC based systems, for example X-Y tables, have already defined the operation of the limit switches and FLAG inputs. Hence, your use of these inputs must be coordinated with the system manufacturer's usage.

DAQ Signals

This manual in can no way cover the wide selection of NI-DAQ boards or their signal sets. For the purposes of this chapter we will consider a few general signals found in some form on most National Instruments DAQ boards that can be used to trigger acquisitions and define the DAQ sample rates. More detailed information can be found in the LabVIEW DAQ examples and tutorials and the manual for your DAQ board. All of the examples in this Chapter use the standard LabVIEW examples.

Connections to the DAQ board are best done using one of National Instruments' many terminal blocks or breadboards. The terminals are well labeled and the chances for shorts are limited.

Analog I/O Channels

You may or may not wish to connect PMAC output signals to the analog input channels on your DAQ board. In several of the examples that follow, we connected the servo clock and JEQU signals to Channel 0 and 1 for the purposes of demonstrating what the clocks look like. To configure, test, and operate these inputs refer to the appropriate National Instruments manual.

Trigger and Scan Clock Connections

You can trigger acquisitions from PMAC in many ways. You can use the JEQU signal to start or stop acquisition at a precise position, or you can use general-purpose digital outputs and synchronous M-Variables. The servo clock and/or the JEQU signal can also be used for the scan clock thereby synchronizing the DAQ sample rate with PMAC's primary timekeeper.

E series boards: Connect your start trigger to PFI0/TRIG1, your stop trigger to PFI1/TRIG2, and your scan clock to PFI7/STARTSCAN.

Legacy MIO boards: Connect your start trigger to STARTRIG*, your stop trigger to STOPTRIG, and your scan clock to OUT2. NOTE: You must scan two or more channels when specifying an external scan clock.

Legacy MIO-16X, MIO-16F-5, and MIO-64F-5: The start and stop trigger pin is EXTTRIG*. Connect your scan clock to OUT2. NOTE: You must scan two or more channels when specifying an external scan clock.

Lab/1200 series boards: Connect your start or stop trigger to EXTTRIG. Hardware pre-triggering (start & stop) is not supported. Connect your scan clock to OUTB1.

For triggers and scan clocks on PC-LPM-16, DAQCard-500, and DAQCard-700 you should refer to the appropriate LabVIEW manual. To find the actual pin numbers, refer to your hardware user manual.

PmacDAQMove

There are dozens of approaches to configuring your particular PMAC/DAQ application. You might consider placing status and position monitoring VIs inside your main DAQ polling loop. This requires you to properly organize the configuration and maintenance of PMAC and DAQ device polling. For the purposes of this Chapter, we selected a multi-threaded model consisting of a main VI to control PMAC and self contained DAQ VIs found in the LabVIEW

examples. We made a few modifications to the DAQ examples such as defining defaults for the sample rates and channel configuration to demonstrate the sampling of the Compare-Equal output, Servo clock, and a simple analog signal. This allowed us to create the examples quickly and validate the operation of PMAC in a more demanding execution environment.

PMAC and AT-MI0-16 Signal Connections

The following examples were built using a PMAC-Lite and a National Instruments' AT-MIO-16. The MIO card signals were accessed using a SC-2070 termination card. The various PMAC signals were accessed using various terminal blocks of the proper sizes. See the Hardware Reference Manual for your PMAC for a list of mating connectors.

In our examples, the DAQ triggers are driven by ENC1. There is no reason other encoders or combinations of signals from multiple motors can't be used with simple modifications.

PMAC Signals

JRS232 (10 pin connector)

Pin 8 - Servo clock (SERVO)

Pin 9 - Common

JEQU (10 pin connector)

Pin 1 - ENC1 Compare Equal Output (EQU1)

Pin 10 - Common

JMACH1 (60 pin connector)

Pin 55 - ENC1 Home Flag Input (HMFL1) – Connected to External TTL Clock

Pin 58 – Common

ATMIO Signals

OUT2 – Scan clock – Wired to PMAC SERVO

EXT TRIG* - Start trigger – Wired to PMAC EQU1

CH0 – Wired to signal generator

CH1 – Wired to PMAC EQU1

CH2 – Wired to PMAC SERVO

Commons/Grounds – ATMIO DGND, ATMIO AGND, JEQU and JRS232 commons all wired together

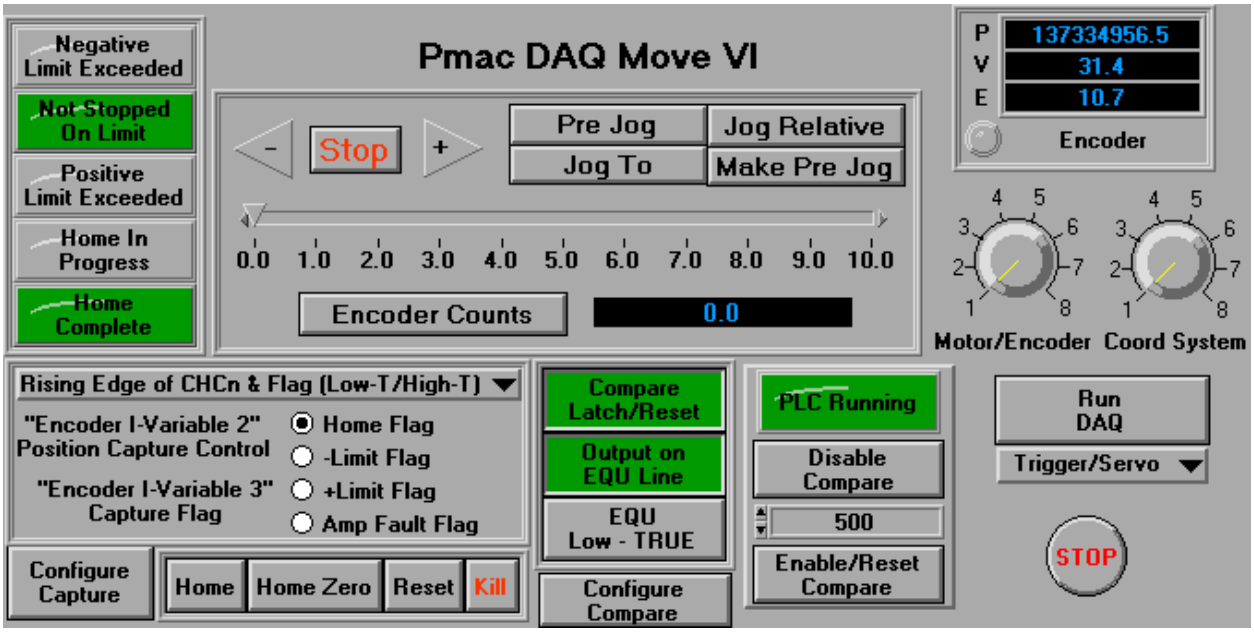
The panel for **PmacDAQMove** is shown below. This VI is comprised of pieces from several of the previous examples. It allows you to home motors, Jog them, configure capture and compare operations, and initiate three different DAQ operations. We will discuss these in a moment.

If you have connected your system in the manner described above or something similar you can begin testing the system by connecting EQU1 to an oscilloscope or by running the VI and using your DAQ board.

When you start the VI select the capture flag configuration and click Configure Capture. Then home the motor. When this has been completed you can set

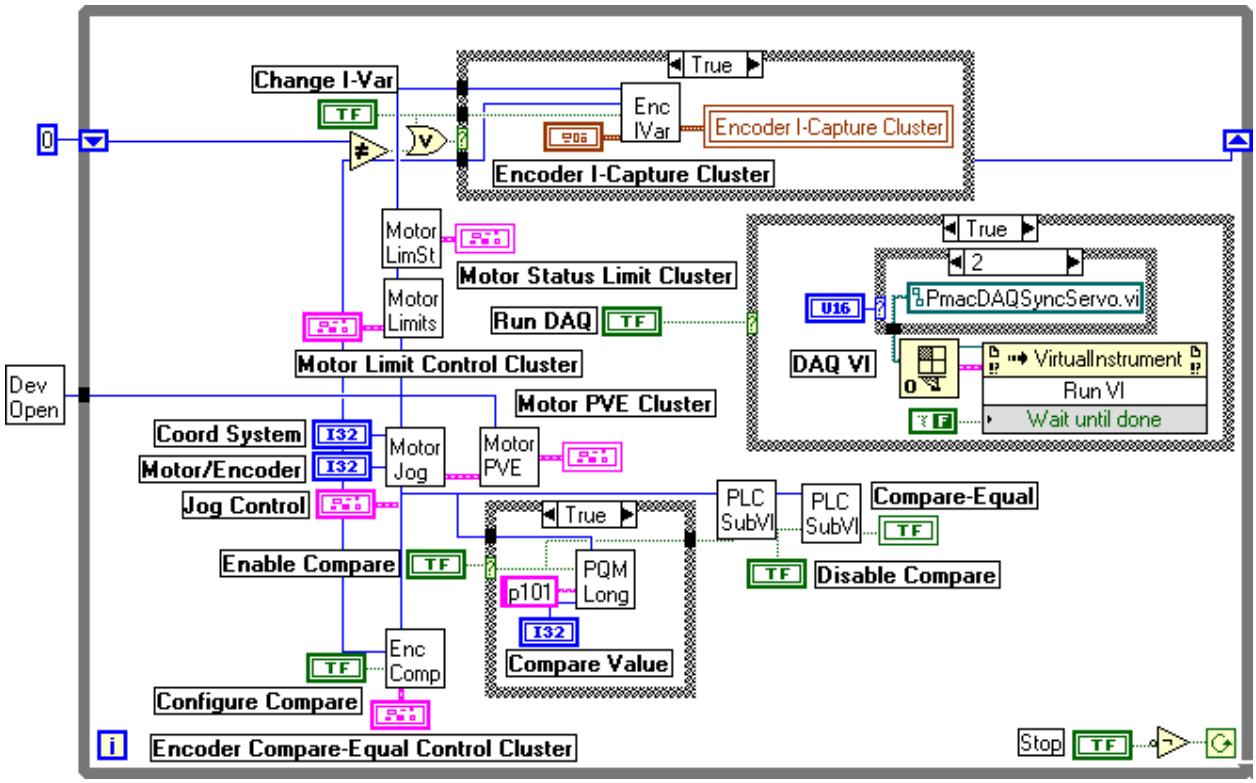
Compare configuration cluster booleans as shown below and click Configure Compare.

This VI uses the encapsulated PLC covered in Chapter 8 to monitor and update the compare-equal operation. The value 500 in the numeric control specifies the pulse generation interval. You can change this if you desire and click Enable/Reset Compare. The Green LED indicates the PLC is running. When you Jog the motor using the jog controls you should see the PVE display update and a pulse train on the scope. You should probably use level triggering on the scope. If you are doing this, the pulse train will have stable intervals with jitter in the actual pulse widths. This is because a background PLC services the encoder. If you need a more stable pulse width, change PLC 19 to PLC 0 in **PmacPosCompGen.pmc** and the reference to PLC 19 in **PmacPosCompSetup.pmc** to PLC 0. The foreground PLC will be serviced more regularly thereby resetting the output in a more deterministic manner.



If you are not using an oscilloscope, you can use the DAQ card to do the same thing. On the far right below the two selector knobs on the panel is a menu ring that allows you to select three different DAQ VIs. Clicking the Run DAQ button starts the selected VI as a separate application thread. These VIs are slightly modified versions of standard LabView examples located in the **Examples\Daq\Anlogin** library supplied with LabVIEW.

The diagram uses the expected pieces from previous examples. The encapsulated PLC that generates the pulses is enabled by the setup PLC and can be disabled by the Disable button. The Run DAQ button uses the Server VI to start the selected PmacDAQ VI as a separate thread.



The three VIs, as named in **\PmacDAQ**, are described below. The name of the original VI in the LabVIEW example library is included in parentheses. We will discuss the DAQ VIs briefly in the following sections with a view to understanding PMAC's signals and their use by the DAQ board.

- **PmacDAQTrigger** – (Cont Acq&Graph (buffered) D-Trig.vi)

  This VI continuously acquires data from one or more analog input channels when a digital start trigger occurs. This is a timed acquisition, meaning that a hardware clock is used to control the acquisition rate for fast and accurate timing. It is a buffered acquisition, meaning that the data are stored in an intermediate memory buffer after they are acquired from the DAQ board. Data are retrieved from that buffer and displayed on the graph.

- **PmacDAQSync** – (Acquire N - Multi-Digital Trig.vi)

  This VI retrieves the specified amount of data from one or more analog input channels each time a digital start trigger, digital stop trigger, or digital start and stop trigger, occur. It shows how to trigger an acquisition multiple times while avoiding the overhead of configuration and buffer allocation each time. This is a timed acquisition, meaning that a hardware clock is used to control the acquisition rate for fast and accurate timing. It is a buffered acquisition, meaning that the data are stored in an intermediate memory buffer after they are acquired from the DAQ board.

- **PmacDAQSyncServo** – (Cont Acq&Graph ExtScanClk D-Trig.vi)

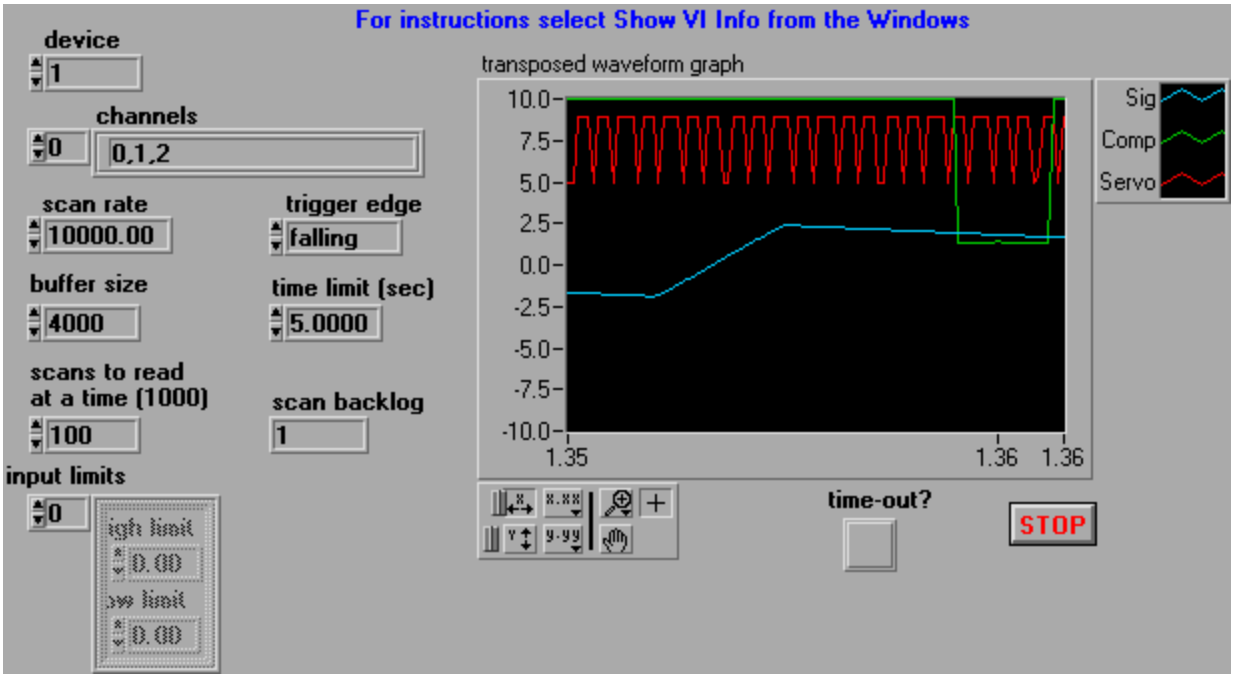
  This VI retrieves the specified amount of data from one or more analog input channels when a digital start trigger, digital stop trigger, or digital start and stop trigger, occur. This VI uses an external scan clock to continually retrieve data from one or more analog input channels. This VI

will only work on devices where you can externally connect a scan clock signal. It is a buffered acquisition, meaning that the data are stored in an intermediate memory buffer after they are acquired from the DAQ board.

Single Trigger DAQ

PmacDAQTrigger is a LabVIEW example that waits for the external trigger supplied by EQU1 to begin asynchronous acquisition at the specified sample rate. In our version of the example, channels 0, 1, and 2 are sampled at 10KHz. The waveform chart in the panel below shows a triangle signal, the EQU trigger pulse, and the servo clock. You can run this VI by selecting Simple DAQ in the menu ring, and clicking the Run DAQ button on **PmacDAQMove** AFTER the PLC is configured and enabled AND the motor is jogging. If the motor is not jogging, the VI will wait 5 seconds for the trigger and then time-out. Good luck starting this VI then starting the motor.

The default servo clock has a 442µS (2262 Hz) update rate. As can be seen in the chart the EQU signal is active LOW. If you check the **PmacDAQMove** panel shown above, you will see that indeed the operation is configured for EQU Low – TRUE. Hence, the initial trigger that started this acquisition started on the falling edge or leading edge of this pulse.

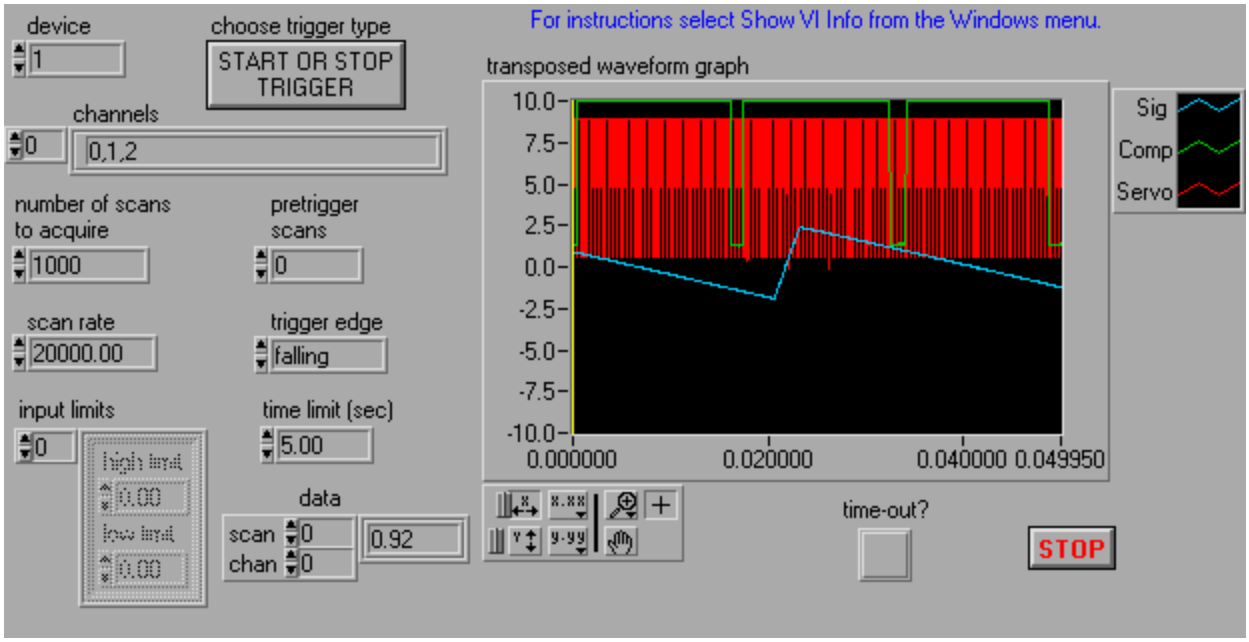


Multi-Trigger DAQ

PmacDAQSync is a LabVIEW example that performs repeated acquisitions synchronized by the external trigger supplied by EQU1 at the specified sample rate. In our default version of the example, channels 0, 1, and 2 are sampled at 20KHz. The waveform chart in the panel below shows a triangle signal, the EQU trigger pulse, and the servo clock. You can run the run this VI selecting Trigger Only in the menu ring, and clicking the Run DAQ button on

PmacDAQMove AFTER the PLC is configured and enabled AND the motor is jogging. If the motor is not jogging, the VI will wait 5 seconds for the trigger and then time-out.

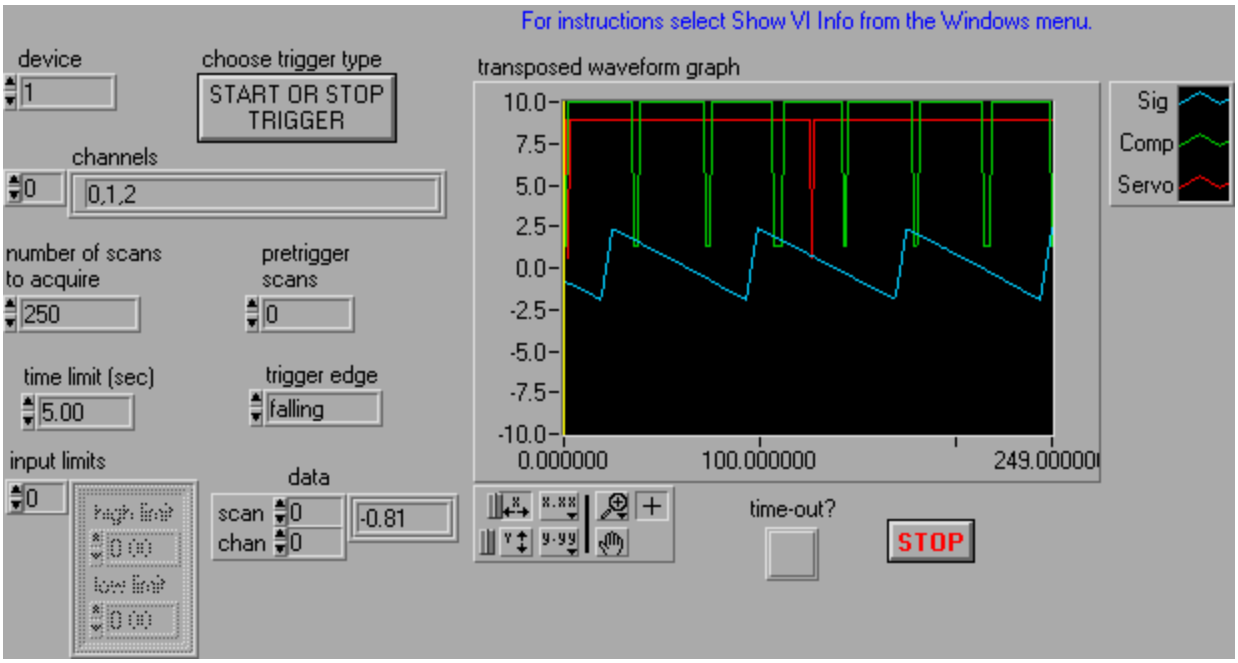
The EQU signal is configured to be active LOW. Hence, the first EQU pulse on the far-left starts with the falling edge of the signal. The default motor Jog rate configured by Ix22 is 32 counts/mS. With a position interval of 500 counts as configured by **PmacDAQMove** this results in a pulse every 15.625 mS. Sure enough, the next pulse occurs right around 15 mS in the chart.



Multi-Trigger DAQ with Servo Clock Sampling

PmacDAQSyncServo is a LabVIEW example that performs repeated acquisitions synchronized by the external trigger supplied by EQU1 using PMAC's servo clock as the scan clock. In our default version of the example channels 0, 1, and 2 will be sampled at PMAC's default servo clock of 2262 Hz. The waveform chart in the panel below shows a triangle signal, the EQU trigger pulse, and the servo clock. You can run this VI by selecting Trigger/Servo in the menu ring, and clicking the Run DAQ button on **PmacDAQMove** AFTER the PLC is configured and enabled AND the motor is jogging. If the motor is not jogging, the VI will wait 5 seconds for the trigger and then time-out.

The EQU signal is configured to be active LOW. You can see the jitter in the EQU signal more clearly in this example. The analog signal is now sampled synchronously with the servo clock. You will note that the servo clock trace in the chart is even caught once on CH2.



Further Sampling Options

The three examples presented here demonstrate that you have many options for triggering and controlling the sampling of your data.

- If you sample the servo clock or the EQU signals along with your data, you have in essence a time code synchronized with your data.
- If you perform a gather on one of PMAC's position or velocity registers while using position compare intervals you can relate your sampled data with precise cycle by cycle motor positions or other motion characteristic.

Using these approaches, you can achieve servo accurate positions for every sample of your data. With LabVIEW's analysis tools you can perform detailed data reductions relating position, velocity, and physical measurements.

Other Interface Options

There is no reason you can't use your DAQ boards DIO to control PMAC's MIO inputs and vice versa. Multipurpose DAQ boards having timers can be used to generate position capture triggers as can the board's D/A capabilities. Other possibilities include the use of the timers to generate time-base control for PMAC.

Although not covered here you can use the sample PMAC generated clocks and signals to trigger and clock bench instruments that you communicate with using GPIB.

Chapter 10 - PComm32 Code Interface Nodes

Basics

This Chapter documents a basic framework for developing LabVIEW Code Interface Nodes (CINs) using Microsoft Visual C++. This topic is important if you

- Desire more sophisticated control over a VIs implementation
- Intend to understand and modify the **PmacDPR** collection of VIs for accessing Dual Ported RAM
- Use PMAC's interrupt capabilities

PMACPanel primarily interfaces to PComm32 using LabVIEW Call Library VIs to access specific PComm32 functions. In some instances, your need for increased speed, sophisticated manipulation of PMAC, or the number of calls to PComm32 begins to create a nasty mess of Call Library VIs that becomes unmanageable.

In this Chapter, we introduce a basic **PmacCIN** VI comprised of a VI, C/C++ source file, Microsoft Visual C++ workspace, and project file. We show how to create the C file, modify the existing workspace and project file, compile the source file, and load the object file into the VI's Code Interface Node. LabVIEW 5.0 makes the process very easy - IF - you follow some simple procedures.

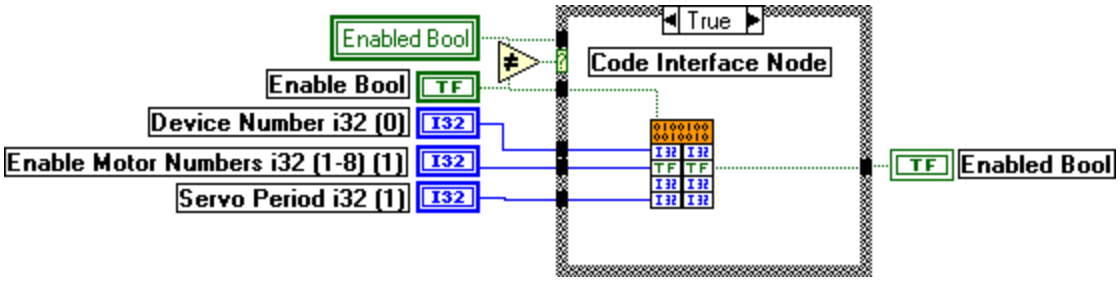
In the Chapter 11 we make extensive use of CINs to handle PMAC's DPR. So if you intend to really understand what can be done and how to do it then this chapter is important to you.

LabVIEW Code Interface Node Basics

What is a CIN?

Code Interface Nodes are VIs that call code written in C, directly from a block diagram. Many LabVIEW aficionados dislike CINs. However, there are instances where the logic required to implement an operation is much simpler to

specify in C than LabVIEW's G. Furthermore, there are instances where the need for efficiency and speed suggest the use of CINs. Accessing PMAC DPR has both requirements. CINs can accept any LabVIEW data type including clusters and arrays as an input or output. The following illustration shows a CIN in a simple diagram.



LabVIEW provides several routines that make working with G data types easier. These routines support memory allocation, file manipulation, and data type conversion. Detailed documentation on these topics can be found in the *LabVIEW Code Interface Reference Manual*.

Using a CIN with PComm32

Appendix A contains an application note available from www.natinst.com fully defining the process for creating a CIN and configuring Microsoft Visual C++ to edit, compile, and link the source code. It's a bit involved but important information. The next section details the configuration information required to add PComm32 support to the basic CIN described in the appendix.

Setting up a PMACPanel CIN Configuration

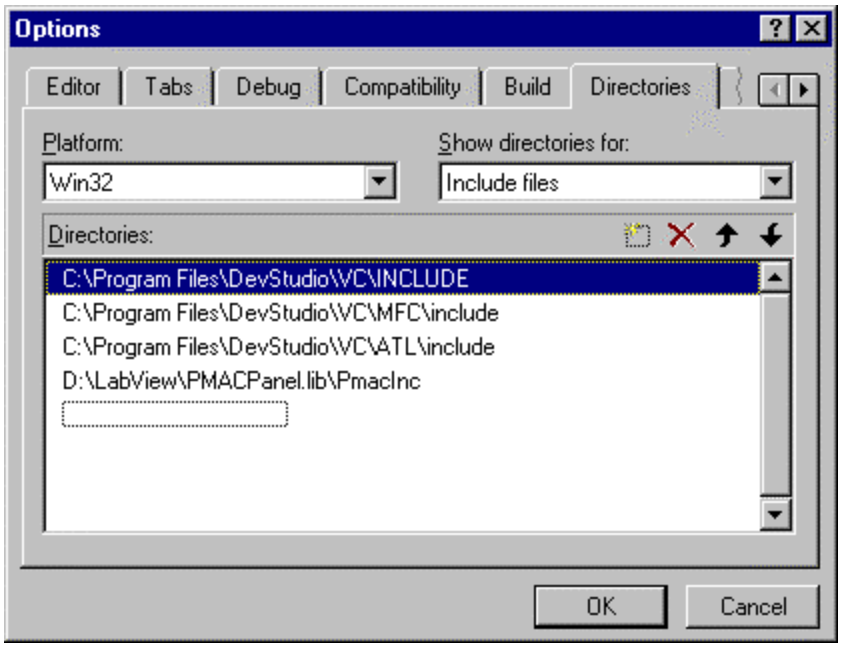
There are two ways to create a project file for the CIN source code created by the LabVIEW. The information presented next is of general importance and leads to a much easier way to develop CINs for PMACPanel.

Adding PComm32 Include Path

To access PComm32 you need to add the following directories to the Visual C++ development environment by selecting **Tools»Options** to display a tab dialog. Click the Directories tab and select Include files in the Select directories for drop down menu. If you double click the outlined box in the directory list a dialog box appears allowing you to select a directory for the include path. The dialog shown below already includes the directory

D:LabView\PMACPanel.lib\PmacInc

If you happen to have PComm32 installed you can use this directory. The result should look something like the following.



Adding Pmac.lib to Project

To link the CIN to produce the lsb code resource you need to include Pmac.lib located in **PMACPanel.lib\PmacInc** or your PComm32 installation directory. You can do by selecting '**Project>>Add To Project>>Files**' and selecting Pmac.lib in the locations noted. You can also add the file and add the path to the Library files selection in the Tool options the same way you added the include path.

Configuring the IDE

Appendix A has detailed instructions one the steps required to configure a project so that it will successfully compile a C file into a loadable code resource for the CIN. This is a bothersome process if you do it a number of times. It may be necessary for you to do this the first time you create and compile a CIN. After that you can use the techniques detailed next to duplicate the project file.

The Easy Way to Add New Projects

The easy way to create new CIN projects is to create a copy of the **PmacCIN.dsp** project file and workspace **PmacCIN.dsw** located in **\PmacCIN** and modify them using notepad or Microsoft Word. Microsoft doesn't recommend this but it saves a lot of configuring when you create a new CIN. If you have 10 or 12 CINs you will get very tired of configuring and managing all the project files.

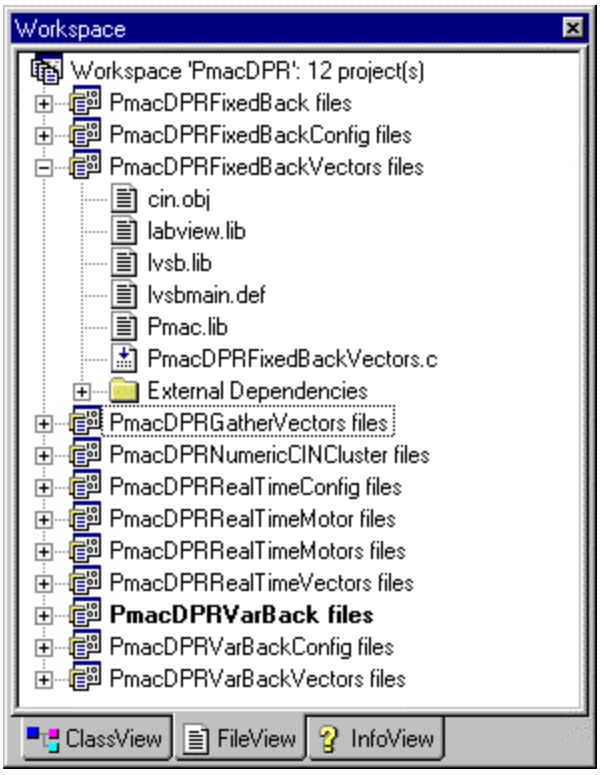
The project file has the keyword **PmacCINBase** used 21 times in it. The workspace has the keyword **PmacCINBase** used once. Copy the files to a new directory, give them new names such as **PmacMyCustomCIN.dsp** and **PmacMyCustomCIN.dsw,** and replace all references to **PmacCINBase** with **PmacMyCustomCIN.** You can then open the new workspace in Visual C++

and compile your project using the LabVIEW created C source file named **PmacMyCustomCIN.c**.

If your installation is different from that contained in **PmacCIN,** try to edit the dsp or dsw files. If there are too many changes, create a Microsoft Visual C++ environment from scratch as outlined in Appendix A, add the paths and libraries for PMAC, and get one project to work. From this point on you can copy your dsp, edit it, and insert it into an existing workspace. It takes a little work the first time, but adding new CINs is very easy after that.

Multiple CIN Projects in a Workspace

Managing multiple CIN projects gets troublesome quickly if each CIN has its own project and workspace. If you navigate your way to the **PmacDPR** directory and open the Microsoft Visual C++ workspace named **PmacDPR.dsw,** you will see that the workspace, shown below, has 12 projects in it. **PmacDPRVarBack** is the currently active project and will be compiled when the Build command is selected. This figure is also instructive in that the project **PmacDPRFixedBackVectors** is open. It shows that any LabVIEW CIN project requires cin.obj, labview.lib, lvsb.lib, and lvsbmain.def. For PMACPanel Pmac.lib is also required. The file **PmacDPRFixedBackVectors.c** is the C source file created by LabVIEW for the CIN node and contains the actual code to accomplish the desired task.



Creating a CIN C-Stub for PComm32

After placing the CIN VI in your diagram, wiring your inputs, and creating the C-source file you get to edit the source. The following code was created for the Code Interface Node in the VI **PmacCIN**.

```
/* CIN source file */

#include "extcode.h"

CIN MgErr CINRun(int32 *Device_Number_i32_0_,
            LVBoolean *Enable_Bool,
            int32 *Enable_Motor_Numbers_i32_1_8_1_,
            int32 *Servo_Period_i32_1_);

CIN MgErr CINRun(int32 *Device_Number_i32_0_,
            LVBoolean  *Enable_Bool,
            int32 *Enable_Motor_Numbers_i32_1_8_1_,
            int32 *Servo_Period_i32_1_) {

      /* ENTER YOUR CODE HERE */

      return noErr;
}
```

To access PComm32 capabilities you need to add the line

```
#include <pmacu.h>
```

You can then utilize all of PComm32's capabilities. In the Chapter that follows we make extensive use of CINs for the implementation of VIs to access PMAC's DPR.

Chapter 11 - DPR - Dual Ported RAM

Basics

Every collection of VIs presented so far uses ASCII command strings to communicate between PMAC and the host. This is independent of whether the actual transfer between the host and PMAC takes place over a serial port, the bus, or DPR. The parsing, formatting, handling, and interpretation of the commands is responsible for most of the time required for communication – even communication that takes place using Dual Ported RAM (DPR).

Dual Ported RAM provides seven other mechanisms for the transfer of limited and specific sets of numeric data between the host and PMAC that requires far less handling. This results in much faster transfers that may be advantageous in your application.

These mechanisms are:

1. **Fixed Real Time Data Buffer** – Automatic copying of limited servo data to the host at a specified servo rate
2. **Fixed background Data Buffer** – Automatic copying of limited motion program data to the host on an as-requested basis
3. **Variable Background Data Buffer** – Automatic copying of user specified data to the host
4. **General Numeric Access** – Bi-directional transfer of numeric data between the host and PMAC using any DPR addresses not dedicated to another operation
5. **Control Panel** – Emulation of PMAC's HW control panel
6. **Binary Rotary Buffer** – Execution of motion programs loaded by the host on an as-requested basis
7. **Real Time Data Gathering** – Automatic copying of user specified data to the host

Of these seven mechanisms 1 through 4 are completely supported by the **PmacDPR** collection of VIs. Mechanism 5 is not really required in that the

same capabilities, albeit slightly slower, are provided using the existing VIs. Mechanism 6 is way beyond the requirements of anything a developer should attempt with PMACPanel. PMAC users generally do not use mechanism 7.

In this Chapter we introduce four collections of VIs in **PmacDPR** that provide the capabilities required for mechanisms 1-4. These are:

- **PmacDPRFixedBack** – Fixed Background data buffer
- **PmacDPRNumeric** – General numeric access to unallocated DPR memory
- **PmacDPRRealTime** – Fixed Real Time data buffer
- **PmacDPRVarBack** – Variable Background data buffer

In each of the four collections contains

- A configuration VI to enable and configure the operation
- VIs to read or write the data using convenient clusters and types
- VIs to buffer multiple data samples into vectors for charting and analysis.
- Examples of all capabilities

Several indicators and controls are provided to handle the data in easy to use clusters. In many instances, previously introduced concepts and clusters are used in ways that tightly integrate the new capabilities available through DPR into PMACPanel's familiar architecture.

We are not going to cover the CINs for all of the VIs in these collections. Once the structure of a configuration, fetch, and vector VI are understood for one of the collections the others will be readily duplicated. You may well find that your applications require a slightly different architecture than that presented here. If so, you can modify the existing CINs to suit your particular needs.

Required Background Understanding

Before reading this material, you must have some familiarity with PMAC's DPR capabilities. This can be found in the *PMAC User Manual*, *PMAC Software Reference Manual*, and PMAC Dual Ported RAM manual. The architecture of the CINs to support the VIs is heavily influenced by the PComm32 DPR API so having that manual available is also necessary. In particular the Chapter PComm32 DPR Features should be read before proceeding.

General Architecture Notes

PMACPanel's DPR support is designed to be simple and extensible. There is not a lot of error checking. The basic assumption is that you have a DPR card with your PMAC. **PmacDPR** doesn't automatically check for this nor does it automatically enable or disable itself. As you inspect the examples, you will see that each mechanism has a configuration VI. As you develop your applications you might want to move these into your **PmacDevOpen** VI so that opening PMAC also enables DPR the way you desire.

PmacDPRRealTime

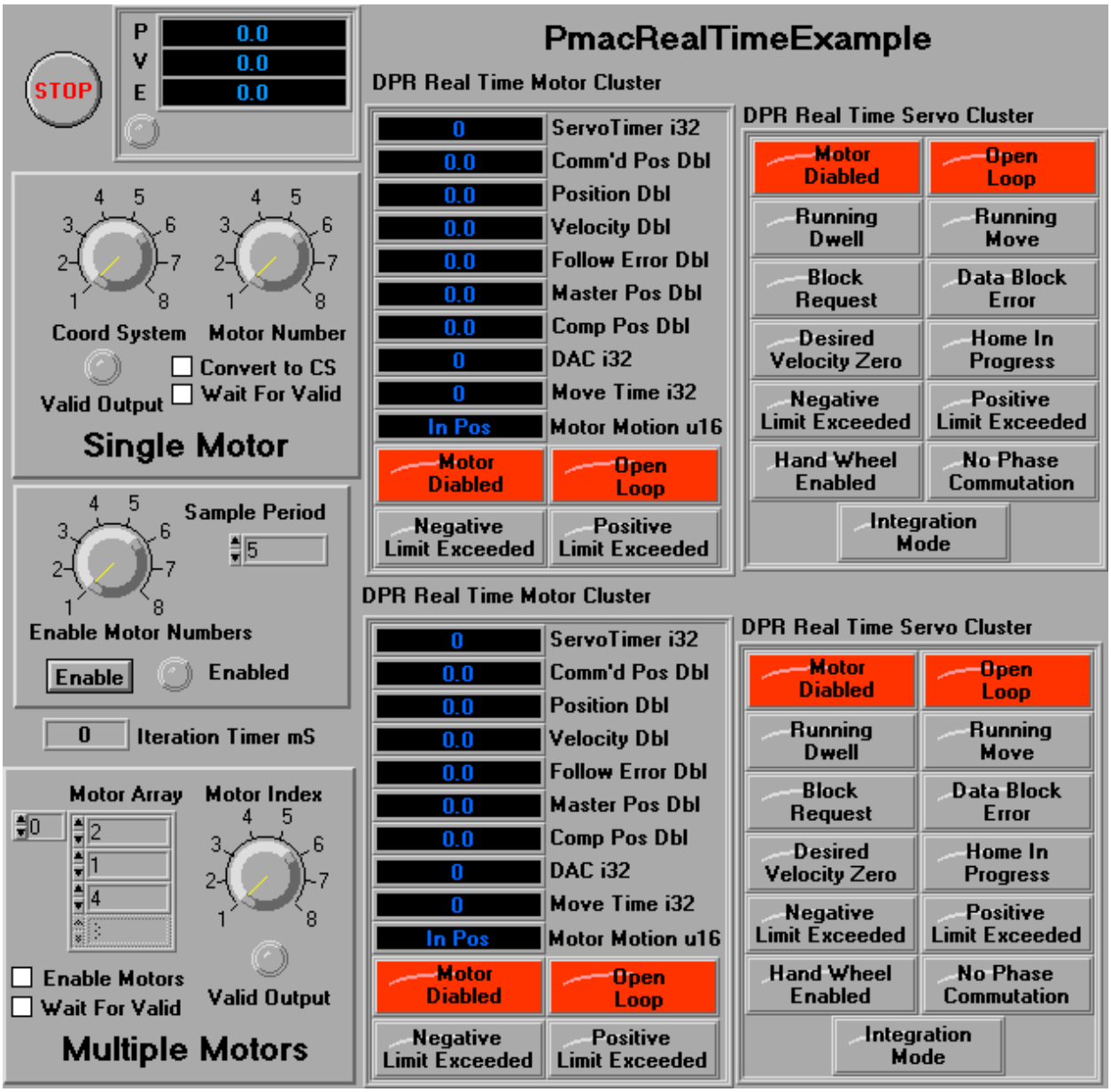
PMAC's Real Time data buffer mechanism automatically copies 27 selected Motor Calculation Registers from their native PMAC locations to DPR locations at a specified servo-cycle sample rate. PComm32 supplies a set of routines to read these values from DPR and convert them into legitimate Intel formats. This process requires some handshaking between PMAC and the host to avoid collisions when accessing DPR from the host.

To minimize your work and simplify the interface, the **PmacDPRRealTime** collection bundles the 23 most useful items them into clusters. This ensures that all data items are gathered during the same servo cycle. It also prevents you from having to wire 27 VIs.

PmacDPRRealTimeExample

The following example demonstrates three **PmacDPRRealTime** VIs. One to configure and enable the operation of Real Time data buffering and two to fetch the data. **PmacDPRRealTimeMotor** collects the data for a single motor. **PmacDPRRealTimeMotors** collects the data for a set of motors. Grouping the fetch of data for multiple motors into a single VI ensures that the data for each motor will be from the same servo cycle.

The panel for the example is shown below. The panel demonstrates the operation for a single motor on the top and multiple motors on the bottom. On the right are two clusters for displaying the data fetched from DPR. On the right are controls for selecting which motor or motors and controlling the fetch from DPR. On the left in between the two is a small block of controls to enable DPR Real Time data buffering.



To execute the example you should select how many motors you want PMAC to copy to DPR using the knob labeled Enable Motor Numbers. Sample Period is the number of servo cycles between copies to DPR. The default value of five indicates that PMAC will update the Real Time data buffer every five servo cycles. With a default servo rate of 2.2 kHz, this corresponds to a 400 Hz sample rate. If you click Enable, PMAC's Real Time data buffer will be enabled. You should immediately see updates taking place in the DPR Real Time Motor Cluster and DPR Real Time Servo Cluster on top. Most noticeably, you will see the Servo Timer increment rapidly reflecting the servo time the sample was taken. If you enabled four motors, you can use the Motor Number knob in the Single Motor box to fetch and display the data for the corresponding motor.

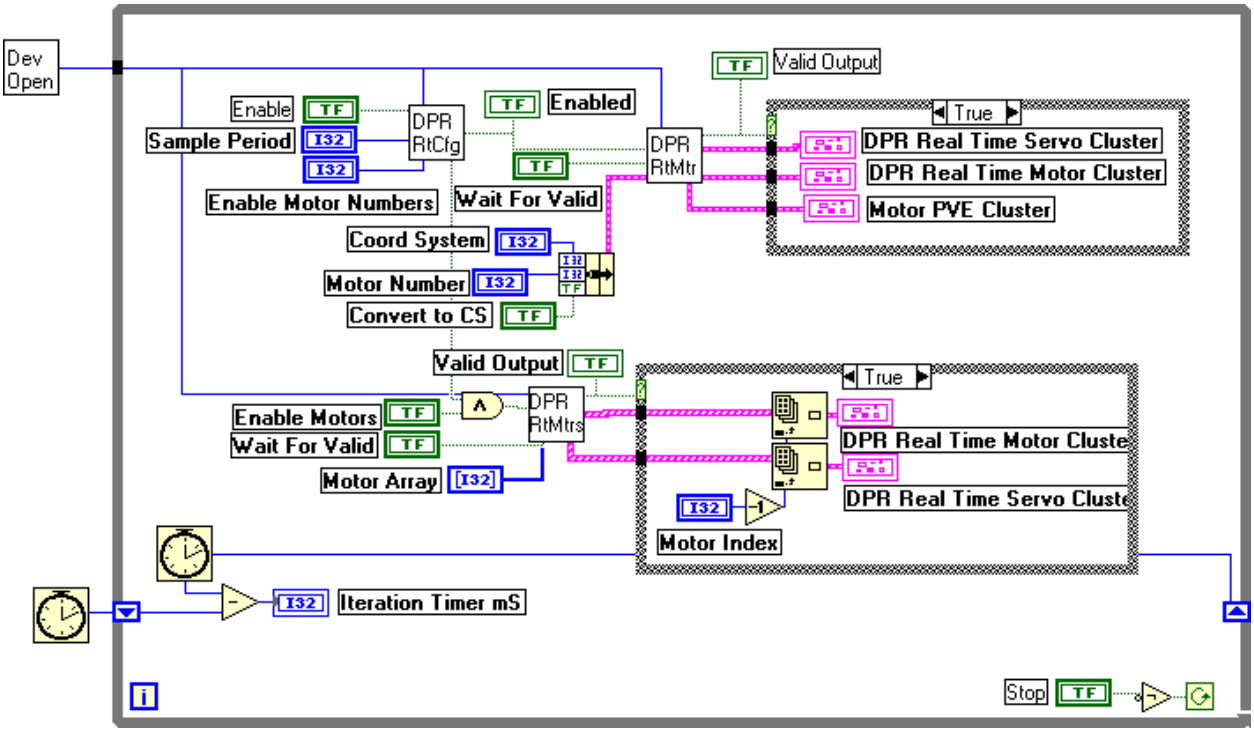
You will note the familiar **PmacMotorPVE** cluster on the top left displays the PVE as a subset of the data contained in the DPR Real Time Motor Cluster. If

you check the Convert to CS box the same conversion to coordinate system units covered in several earlier chapters is applied to the PVE data gathered from DPR.

Below this collection of controls is an indicator labeled Iteration Timer that display the time in mS for each loop iteration. On average the fetch and display update of DPR Real Time data for a single motor takes 1-2mS.

You will notice the Green LED in the Single Motor box flickering on and off. When PMAC copies data to DPR, it sets a 'Busy Bit' indicating that it is accessing DPR. During this time, the host, running this VI, cannot access DPR. To avoid possible problems the VI simply indicates that it did not perform a successful fetch. If you check the box labeled 'Wait For Valid', the VI will continue placing calls to the associated CIN until it performs a successful read.

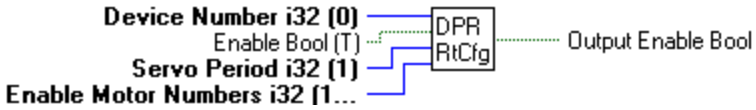
The demonstration on the bottom of the panel is for multiple motors. To fetch the data for multiple motors you supply an array of motor numbers and enable the fetch. The VI then fetches the data for the specified motors and returns an array of clusters. The time required for this multiple motor fetch is on the order of 1-2 mS. In this example, you can select a Motor Index for display. As shown a Motor Index of 1 displays the data for Motor 2.



The diagram for the example requires three VIs. One to configure the operation, one to fetch the data for a single motor and one to fetch the data for multiple motors. Case statements are used to control the update of the display clusters. The architecture of the **PmacDPR** VIs is a little different from most of those already introduced. Whereas almost all other collections operate in a query response mode that is always enabled these VIs require the enabling of specific capabilities by a configuration VI. Hence, most **PmacDPR** VIs have an enable input that prevents them from querying DPR until it is enabled.

On the top left the **PmacDPRRealTimeConfig** VI requires a Sample Period, Enable Boolean, and an integer indicating how many motors to copy. The enable input is not latched. When it is TRUE the Real Time data buffer is enabled. When it is FALSE the operation is disabled.

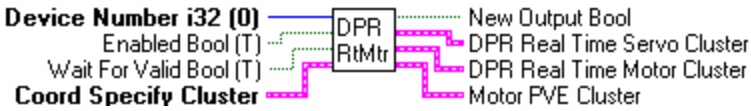
- **PmacDPRRealTimeConfig** - DPR Real Time Fixed buffer is configured to update motor information every Servo Period for all motor between 1 and Enable Motor Numbers when Enable is TRUE. Operation is disabled when Enable is FALSE. Output Enable is TRUE when operation is enabled. The state is maintained by the VI. Operation of DPR Real Time buffers overlaps with DPR Fixed Background operation in that the number of motors enabled must be the same.



PmacDPRRealTimeMotor fetches DPR data for a single motor. This has an optional enable signal, in this case provided by the configuration VI. It also has an optional Wait For Valid input and a Coord Specify Cluster that is used to specify the Motor Number, and standard Coordinate System conversions for the production of the PVE cluster.

- **PmacDPRRealTimeMotor** - Query PMAC DPR for the Real Time Fixed buffer Motor and Servo data. When Enabled is TRUE (the Default state) the data for Motor Number is fetched and used to build DPR Real Time Motor Cluster and DPR Real Time Servo Cluster. Motor PVE Cluster contains data in encoder counts or coordinate system units depending on the state of Coord Specify Cluster. See PmacMotorPVE for details on how this is done.

  A successful query of PMAC's DPR depends on whether PMAC is accessing the memory. If Wait For Valid is TRUE the VI places queries to PMAC until a successful read at which time New Output is TRUE indicating valid output data. If Wait For Valid is FALSE the query may or may not succeed. If the query fails New Output is FALSE and the output clusters contain the data fetched during the last read.

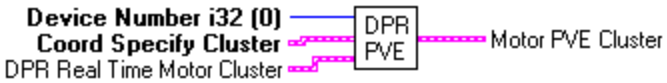


The fetching of multiple motor data by **PmacDPRRealTimeMotors** requires an array of motors and produces an array of clusters. This VI differs from **PmacDPRRealTimeMotor** in that it does not provide Coordinate System conversions. This is not provided because it would require you to assemble an array of Coord Specify Clusters. If you require the Coordinate System conversion of Real Time data for multiple motors you can use the **PmacDPRRealTimePVE** and apply the transformation to the individual cluster elements in the output arrays.

- **PmacDPRRealTimeMotors** - Query PMAC DPR Real Time Fixed buffer Motor and Servo data for the list of motors specified in Motor Number Array. When Enabled is TRUE (the Default state) the data for the specified motors is fetched and used to build an array of DPR Real Time Motor Clusters and DPR Real Time Servo Clusters.

  A successful query of PMAC's DPR depends on whether PMAC is accessing the memory. If Wait For Valid is TRUE the VI places queries to PMAC until a successful read at which time New Output is TRUE indicating valid output data. If Wait For Valid is FALSE the query may or may not succeed. If the query fails New Output is FALSE and the output clusters contain the data fetched during the last read.

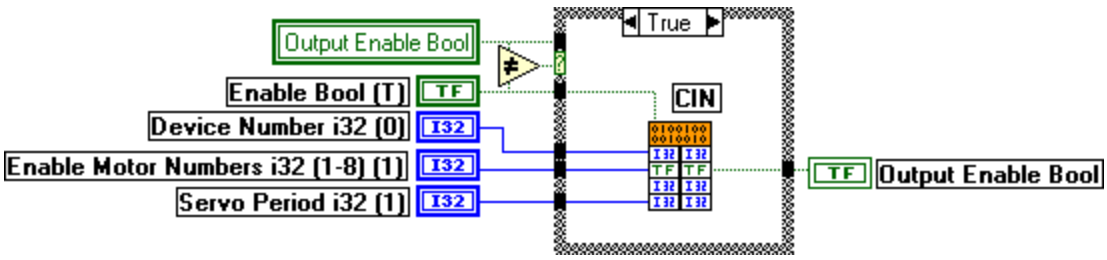
- **PmacDPRRealTimePVE** - Extract position, velocity, and following error from DPR Real Time Motor Cluster assuming Motor Number operating in Coord Number. Assemble the measurements into Motor PVE Cluster. If Convert is TRUE convert the measurements to CS units. Otherwise leave them in encoder counts.



PmacDPRRealTimeConfig CIN

We are not going to cover all **PmacDPR** CINs in the same detail we do here. Once you understand the basics of these, your understanding of the other collections will follow.

The very simple diagram for **PmacDPRRealTimeConfig** is shown below. You will note that the Device Number, Enable Motor Numbers, and Servo Period are all passed to the CIN. Even the Enable is passed. The CIN returns an Output Enable signal that indicates whether the configuration and enable operation succeeded. The only unique characteristic of this VI is the compare operation between the Output Enable and Input Enable. When these values are not equal, the TRUE case executes enabling or disabling the operation as defined by the input Enable.



Many things can be done with a CIN. You should have a copy of the LabVIEW CIN Reference Manual when working with these until you get familiar with how

LabVIEW passes parameters. This is especially true for clusters and arrays. Things get more interesting when looking at the actual C code that implements this CIN. This is shown below.

```
/* * CIN source file */

#include "extcode.h"
#include <pmacu.h>
#include <dprrealt.h>

/* stubs for advanced CIN functions */

UseDefaultCINInit
UseDefaultCINDispose
UseDefaultCINAbort
//UseDefaultCINLoad
UseDefaultCINUnload
UseDefaultCINSave

// -- This a GLOBAL variable!

BOOLEAN Enabled = FALSE;

// --

CIN MgErr CINRun(int32 *Device_Number_i32_0_,
      LVBoolean *Enable_Bool_T_,
      int32 *Enable_Motor_Numbers_i32_1_8_1_,
      int32 *Servo_Period_i32_1_);

CIN MgErr CINRun(int32 *Device_Number_i32_0_,
      LVBoolean *Enable_Bool_T_,
      int32 *Enable_Motor_Numbers_i32_1_8_1_,
      int32 *Servo_Period_i32_1_) {

      /* -- When not currently enabled and Enable_Bool_T_ == LVTRUE
      enable the Fixed buffer for the specified number of motors -- */

      if (!Enabled && *Enable_Bool_T_ == LVTRUE) {

            PmacDPRSetMotors(*Device_Number_i32_0_,
                        *Enable_Motor_Numbers_i32_1_8_1_);
            PmacDPRRealTime(*Device_Number_i32_0_,
                        *Servo_Period_i32_1_,
                        1);
            Enabled = TRUE;
      }

      /* -- When currently enabled and Enable_Bool_T_ == LVFALSE
      disable ALL background operations. -- */

      else if (Enabled && *Enable_Bool_T_ == LVFALSE) {
            PmacDPRRealTime(*Device_Number_i32_0_,
                        *Servo_Period_i32_1_,
                        0);
            Enabled = FALSE;
      }
```

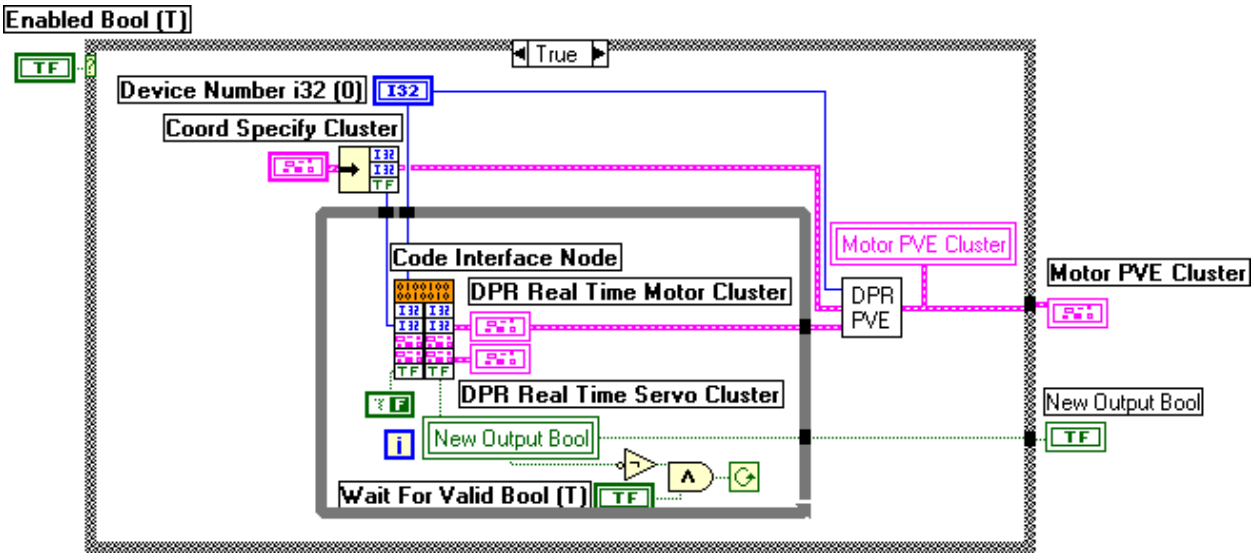
```
      *Enable_Bool_T_ = Enabled;
      return noErr;
}

// When first loaded make sure Enable flag is FALSE

CIN MgErr CINLoad(RsrcFile rf)
{
      Enabled = FALSE;          // Indicate DPR Fixed Real Time disabled

      return noErr;
}
```

This particular CIN has two functions CINLoad and CINRun. LabVIEW creates the function and data type declarations such as clusters and arrays required by CINRun. The function CINRun is called when the VI containing the CIN is executed. CINLoad is executed when the VI is first loaded.

In this sample, you will note that two functions from PComm32 are used. **PmacDPRRealTime** enables and disables the DPR Real Time data buffer and **PmacDPRSetMotors** sets the number of motors to copy. The logic of the **if** statement uses the current enable state of the operation and the desired state passed in by *Enable_Bool_T_ to turn the DPR Real Time data buffer on or off. Notice the type of the Boolean and that most parameters are passed as pointers to data.

PmacDPRRealTimeMotor CIN

The diagram for **PmacDPRRealTimeMotor** is shown below. You will note that the Device Number, Motor Number, and a FALSE Boolean constant are passed to the CIN. The CIN returns two clusters and a Boolean indicating whether the fetch operation succeeded. The CIN is only executed when the input Enabled is TRUE. If Wait For Valid is TRUE, the while loop will execute the CIN until New Output is TRUE thereby waiting until the fetch from DPR succeeds. Repeated requests for DPR data is done by the diagram rather than the CIN code. If the C-code waited in a while loop that was never satisfied you couldn't abort your application.



This VI's CIN code has a simple structure but places many calls to PComm32. This is the reason we selected to implement this VI with a CIN rather than 20 or 30 Call Library Function VIs. To simplify the code we have removed several lines from the actual source to demonstrate the basic ideas.

```
/* * CIN source file */

#include "extcode.h"
#include <pmacu.h>
#include <dprrealt.h>

// Types defined by LabVIEW when the stub was created

typedef struct {
      int32 ServoTimer_i32;
      float64 Comm_d_Pos_Dbl;
      float64 Position_Dbl;
      float64 Velocity_Dbl;
      float64 Follow_Error_Dbl;
      float64 Master_Pos_Dbl;
      float64 Comp_Pos_Dbl;
      int32 DAC_i32;
      int32 Move_Time_i32;
      uInt16 Motor_Motion_u16;
      LVBoolean Motor_Activated;
      LVBoolean Open_Loop;
      LVBoolean Neg_Limit_Exceeded;
      LVBoolean Pos_Limit_Exceeded;
} TD1;

typedef struct {
      LVBoolean Home_In_Progress;
      LVBoolean Block_Request;
      LVBoolean Desired_Velocity_Zero;
      LVBoolean Data_Block_Error;
      LVBoolean Dwell_In_Progress;
      LVBoolean Integration_Mode;
      LVBoolean Running_Move;
      LVBoolean Open_Loop;
      LVBoolean Phased_Motor;
      LVBoolean Hand_Wheel_Enabled;
      LVBoolean Neg_Limit_Exceeded;
      LVBoolean Pos_Limit_Exceeded;
      LVBoolean Motor_Activated;
} TD2;

// --

CIN MgErr CINRun(int32 *Device_Number_i32_0_, int32 *Motor_Number_i32_1_8_1_,
                 TD1 *DPR_Real_Time_Motor_Cluster,
                 TD2 *DPR_Real_Time_Servo_Cluster
                 LVBoolean *ValidData){

      int32 DevNum = *Device_Number_i32_0_;  // Shorter dereferenced name
      int32 MNum = *Motor_Number_i32_1_8_1_ - 1;
      SERVOSTATUS ServoStatus;

      // --  Tell PMAC we're doing our thing
```

```
	PmacDPRSetHostBusyBit(DevNum, 1);

	// -- Check if PMAC is busy doing its thing

	if (!PmacDPRGetPmacBusyBit(DevNum)) {

		// -- Fetch all of the available data

		DPR_Real_Time_Motor_Cluster->ServoTimer_i32 =
			PmacDPRGetServoTimer(DevNum);
		…
		DPR_Real_Time_Motor_Cluster->Motor_Activated =
			PmacDPRMotorEnabled(DevNum, MNum) == 0 ? LVFALSE : LVTRUE;

		// -- ServoStatus – Fetch cluster and then individual items

		ServoStatus = PmacDPRMotorServoStatus(DevNum, MNum);
		DPR_Real_Time_Servo_Cluster->Home_In_Progress =
			ServoStatus.home_search == 0 ? LVFALSE : LVTRUE;
		…
		// --

		*ValidData = LVTRUE;		// New data for caller
	}
	else {
		*ValidData = LVFALSE;		// Sorry - no new data
	}

	PmacDPRSetHostBusyBit(DevNum, 0);// PMAC can do its thing
	return noErr;
}
```

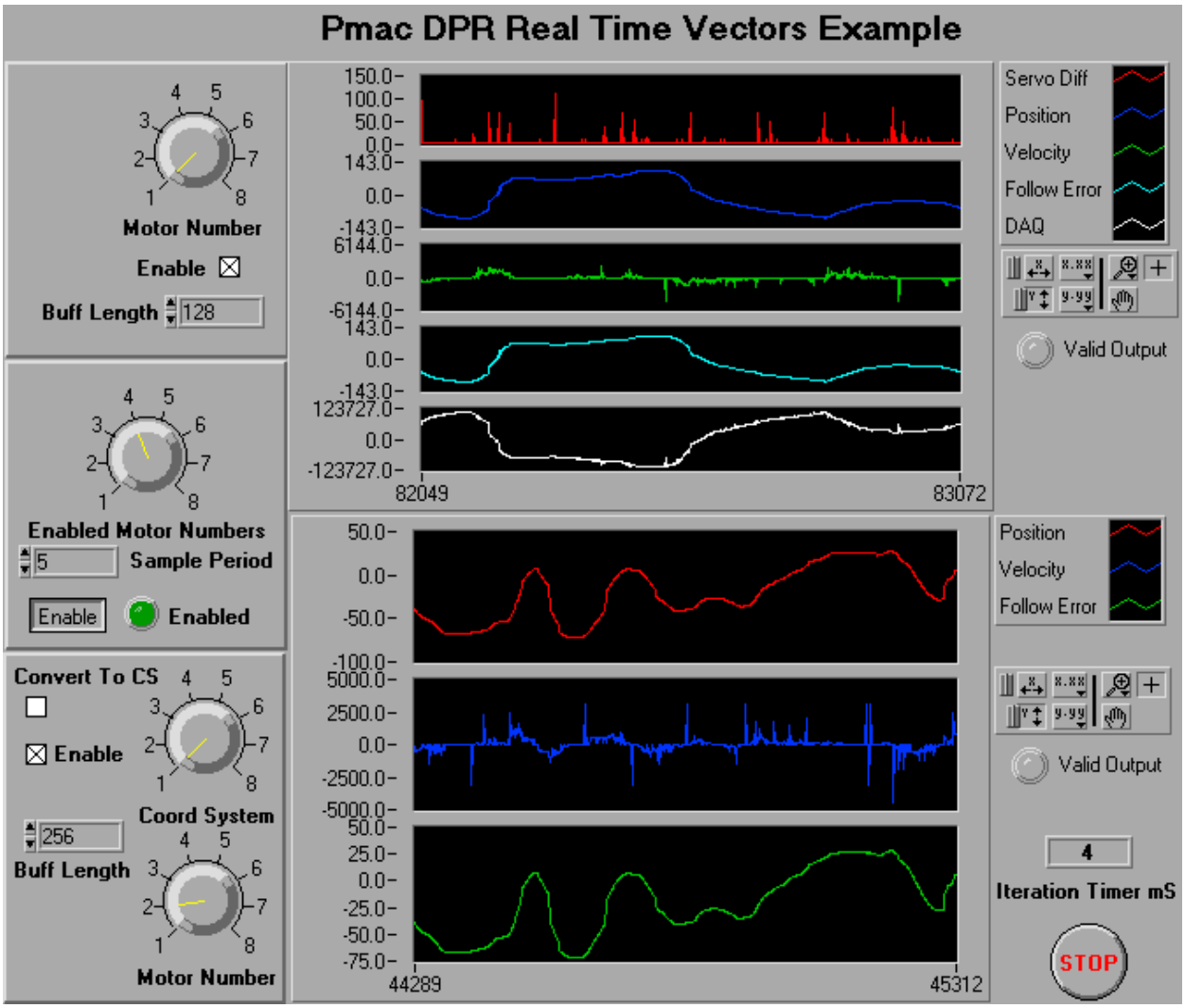
If you check the *PComm32 Reference Manual* you will see that the checking and setting of the DPR Busy Bit is required by PMAC. When the unfilled CIN node stub was created LabVIEW generously declared the CINRun parameter order, names, and data types. CINRun's job is to fetch DPR data from PMAC using the PComm32 functions and fill the LabVIEW data types passed by the caller with the data. It is actually very simple. The parameters DPR_Real_Time_Servo_Cluster and DPR_Real_Time_Motor_Cluster are pointers to the data types provided by LabVIEW. Calls are placed to PComm32 and data of the proper type is assigned to the members of the data types. There is one catch here. You will note that Booleans returned by PComm32 are converted to LVTRUE and LVFALSE before being assigned to the members of the clusters. This is precaution that avoids possible mismatches in data types.

PmacDPRRealTimeVectorExample

The following example demonstrates a very powerful **PmacDPR** technique that takes multiple samples over time. This is done by placing repeated CIN that build vectors for the desired items. This creates a simpler and faster VI diagram because building the vector is done by the CIN and the data is returned by the CIN only when a vector of a specified length is built.

The panel for the example is shown below. The panel demonstrates the fetching of vectors for the purposes of driving a real-time chart. On the top left is a knob for selecting a motor and specifying the number of samples to accumulate before updating the chart. On the bottom the same operations are performed but motor

items such as Position, Velocity, and Following Error are converted to CS units in the standard manner defined for **PmacMotors**. On the left between the two sets of controls is a small block of controls to enable DPR Real Time data buffering.



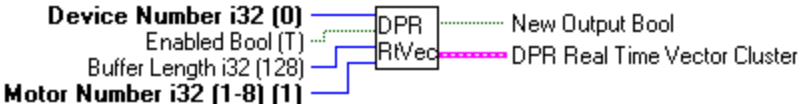
To execute the example you should select how many motors you want PMAC to copy to DPR using the knob labeled Enable Motor Numbers. Sample Period defines the servo-sampling interval. You can then click Enable to start the DPR Real Time data buffer. As with **PmacDPRRealTimeExamp,** you can select a motor to fetch and check the Enable box for the top or bottom chart.

The **PmacDPRRealTimeVectors** VI that actually processes the request for a fetch does a little more book keeping and buffers the data in arrays prior to passing it back to the caller. It returns a **PmacDPRRealTimeVectors** cluster from which the desired items can be selected and plotted as shown here. This cluster differs from **PmacDPRRealTimeMotor** and **PmacDPRRealTimeServo.** Many of the items in the clusters would not generally be of interest in a time-vector. If you desire these, you can modify **PmacDPRRealTimeVectors** to support them.

The demonstration on the bottom panel is different from that on top in that it has a longer buffer (256 samples vs. 128 samples) and a CS transformation can be applied.

- **PmacDPRRealTimeVectors** - Query PMAC DPR for the Real Time Fixed buffer Motor and Servo data. When Enabled is TRUE (the Default state) the data for Motor Number is fetched and used to build DPR Real Time Motor Cluster and DPR Real Time Servo Cluster. Motor PVE Cluster contains data in encoder counts or coordinate system units depending on the state of Coord Specify Cluster. See PmacMotorPVE for details on how this is done.

  A successful query of PMAC's DPR depends on whether PMAC is accessing the memory. If Wait For Valid is TRUE the VI places queries to PMAC until a successful read at which time New Output is TRUE indicating valid output data. If Wait For Valid is FALSE the query may or may not succeed. If the query fails New Output is FALSE and the output clusters contain the data fetched during the last read.



If you enable both portions of the example, you should insure that each requests the data for a different motor. We will discuss the reasons for this in the next section.

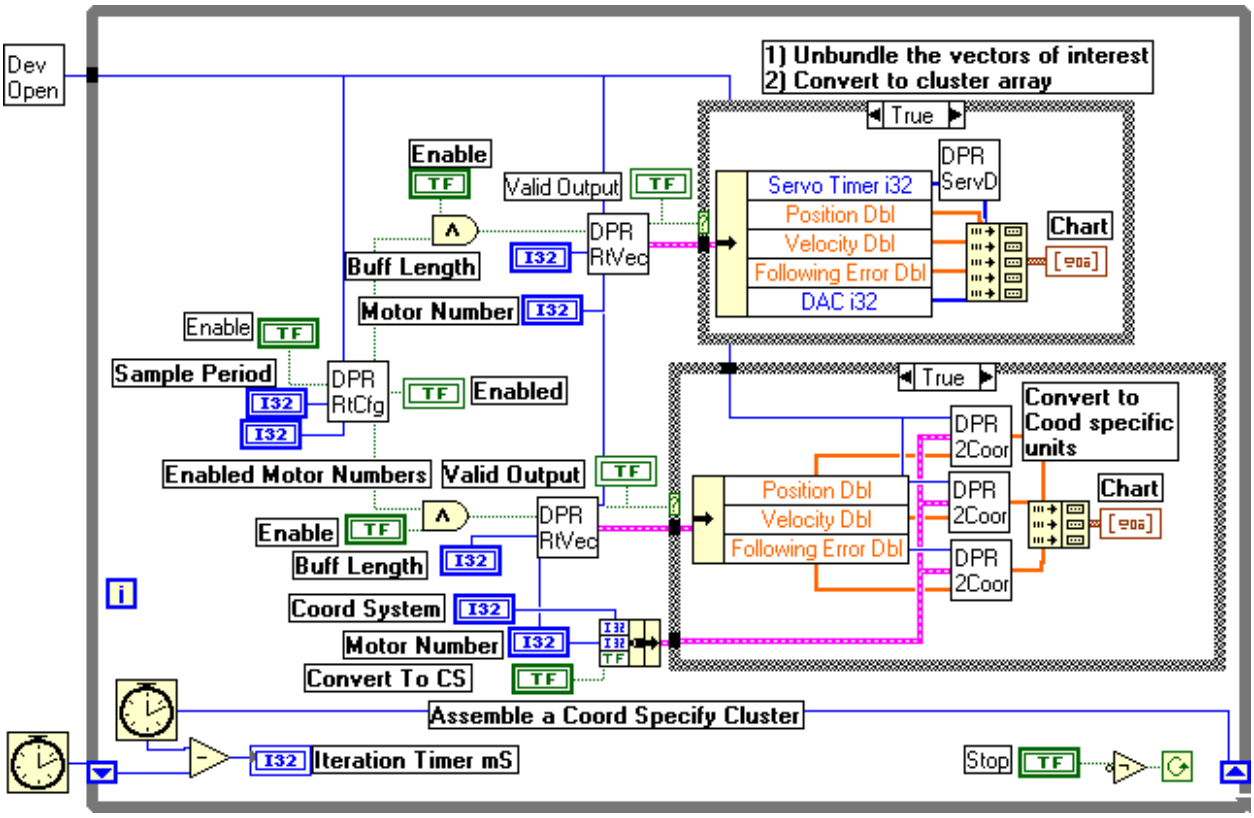
Servo Accurate Sampling

There is an important issue regarding the **PmacDPR** Vector VIs that must be understood to avoid misconceptions. PMAC is a very fast real-time controller that generates more data than could possibly be used in any given application. In Chapter 5, we discussed the **PmacGather** collection of ICVs that utilized PMAC's data gathering capabilities. The gather facility gathers and buffers a specified set of items at a specified servo rate using PMAC memory. The gathered data can be transferred to the host later for decoding and use. The **PmacDPR** Vector VI collects the data in host memory on an as-it-gets-there basis. Samples will be missed when your application is busy with other operations. This can be seen in the example's Red chart strip of the Servo Timer difference. This strip chart is the difference between successive Servo Timer samples and reflects the jitter in the sampling. Most samples are taken every five servo cycles in this example. When a complete buffer is accumulated, passed back to the VI, and then used to update the chart the sampling interval experiences a blip of approximately 75 mS. You will even notice 5 and 10 mS blips in between the major buffer updates.

From a practical point of view, the Servo Timer vector definitively identifies the time each sample was taken. This can be used to resample the other data vectors, or handled however you choose. Until Delta Tau includes DPR data gathering in PComm32 you should use regular data gathering if you absolutely require servo accurate sampling. This form of gathering does not support strip charting on a continuous basis.

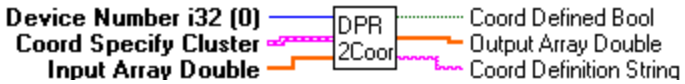
As can be seen in this panel, the interval required to process both fetches is between two and four mS without the update of the charts. When the vector(s) have been accumulated the updating of the charts requires between 75 and 100 mS.

The diagram for the example is very similar to that of **PmacDPRRealTimeExamp**. The configuration VI is the same. The Vector VI **PmacDPRRealTimeVectors** is similar to **PmacDPRRealTimeMotors** in that it takes an Enable input. It also has a Buffer Length input. Rather than a Coord Specify Cluster, you simply provide a Motor Number. The decision to do this is based on the view that whereas single Servo sample data for a motor might be used to drive a PVE type of panel cluster this is not true for a Vector operation. You should note that the update of the chart is wrapped in a case structure. DO NOT use the cluster of arrays unless Valid Output is TRUE. The arrays should have zero length but this may cause problems.



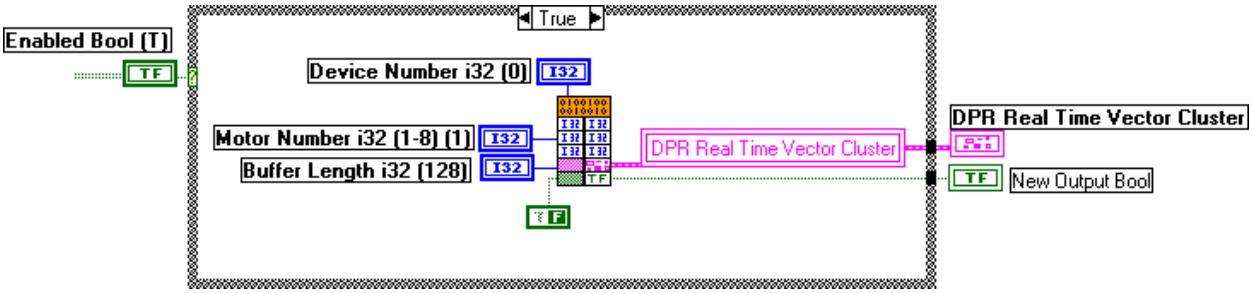
The vector fetch in the bottom half of the diagram unbundles the returned cluster of vectors and performs a CS conversion on the elements of the selected vectors using **PmacDPRMotorVecToCoord.** This is a vector version of the standard **PmacCoord** VIs.

- **PmacDPRMotorVecToCoord** - Coord Specify Cluster specifies a motor within a CS and an attempt to convert Input Array from encoder counts to CS units. If the motor is not defined in the CS no conversion is applied. If the motor is defined and Convert is TRUE Coord Defined is TRUE and Output Array is scaled from encoder counts to CS units. Coord Definition is a string specifying Output Value units as "Encoder" or the CS definition of the motor.



PmacDPRRealTimeVectors CIN

The diagram for **PmacDPRRealTimeVectors** is shown below. You will note that the Device Number, Motor Number, and Buffer Length are all passed to the CIN when Enable is TRUE. The CIN fetches the DPR samples and builds the vectors on every execution of the CIN. 99% of the time the CIN returns a FALSE value for New Output because it still has more samples to accumulate. When it has accumulated Buffer Length of samples it copies them to DPR Real Time Vector Cluster and returns a TRUE for New Output.



The CIN code for this type of VI is a bit more complex than those presented already. It requires a data buffer for storing the accumulated samples and it requires some manipulation of the vectors in the returned cluster. As defined the data buffers are global thereby making them accessible to any reentrant copy of the VI. The topic of handling arrays in CINs is thoroughly covered in the *LabVIEW CIN Reference Manual*. To simplify the code, several lines have been removed from the source to demonstrate the basic ideas.

```
/* * CIN source file */

STANDARD INCLUDES and DEFINITIONS – See actual Source File

// -- i32 vector

typedef struct {
      int32 dimSize;
      int32 Value[1];
} TD2;
typedef TD2 **TD2Hdl;

// -- float64 vector

typedef struct {
      int32 dimSize;
      float64 Value[1];
} TD3;
typedef TD3 **TD3Hdl;

// -- Cluster of arrays

typedef struct {
      TD2Hdl Servo_Timer_i32;
      TD3Hdl Position_Dbl;
      TD3Hdl Velocity_Dbl;
      TD3Hdl Commanded_Pos_Dbl;
```

```
      TD3Hdl Following_Error_Dbl;
      TD3Hdl Master_Position_Dbl;
      TD3Hdl Comp_Pos_Dbl;
      TD2Hdl DAC_i32;
      TD2Hdl Move_Time_i32;
} TD1;

// -- GLOBAL buffers for handling motor data

#define MOTOR_MAX 8
#define BUFFER_MAX 512

int32 BufferCount[MOTOR_MAX];
int32 ServoTimer[MOTOR_MAX][BUFFER_MAX];
int32 DAC[MOTOR_MAX][BUFFER_MAX];
int32 Move_Time[MOTOR_MAX][BUFFER_MAX];

float64 Commanded_Position[MOTOR_MAX][BUFFER_MAX];
float64 Position[MOTOR_MAX][BUFFER_MAX];
float64 Velocity[MOTOR_MAX][BUFFER_MAX];
float64 Following_Error[MOTOR_MAX][BUFFER_MAX];
float64 Master_Position[MOTOR_MAX][BUFFER_MAX];
float64 Compensation_Position[MOTOR_MAX][BUFFER_MAX];

// --

CIN MgErr CINRun(int32 *Device_Number_i32_0_,
            int32 *Motor_Number_i32_1_8_1_,
            int32 *Buffer_Length_i32_128_,
            TD1 *DPR_Real_Time_Vector_Cluster,
            LVBoolean *ValidData) {

      int32 DevNum = *Device_Number_i32_0_;
      int32 MNum = *Motor_Number_i32_1_8_1_ - 1;
      int BCount = BufferCount[MNum];
      int SizeL, SizeD;

      // --  Tell PMAC we're doing our thing

      PmacDPRSetHostBusyBit(DevNum, 1);

      // -- Check if PMAC is busy doing its thing

      if (!PmacDPRGetPmacBusyBit(DevNum)) {

            // -- Get Servo timer
            ServoTimer[MNum][BCount] = PmacDPRGetServoTimer(DevNum);

            /* -- If this is the first element of a new buffer - OR -
            its not the first element of a buffer - AND -
            the current servo timer value is not equal to last measurement
            THEN - fetch the DPR data into the proper buffer elements. -- */

            if ((BCount == 0) || ((BCount > 0) &&
                  (ServoTimer[MNum][BCount-1] != ServoTimer[MNum][BCount]))){

                  Commanded_Position[MNum][BCount] =
                              PmacDPRGetCommandedPos(DevNum, MNum, 1.0);
```

```
                  Position[MNum][BCount] =
                              PmacDPRPosition(DevNum, MNum, 1.0);

                  …
                  BufferCount[MNum]++;
            }
      }

      // -- Check for full buffer then copy to LabVIEW data structs

      if (BufferCount[MNum] == *Buffer_Length_i32_128_) {

            BCount = BufferCount[MNum];

            // -- Resize the arrays in the structure

            NumericArrayResize(iL, 1,
                        (UHandle *) &(DPR_Real_Time_Vector_Cluster-
                        >Servo_Timer_i32), BCount);
            (*DPR_Real_Time_Vector_Cluster->Servo_Timer_i32)->dimSize =
                        BCount;
            …

            // --

            SizeL = BCount * sizeof(int32);
            SizeD = BCount * sizeof(float64);

            memcpy((*DPR_Real_Time_Vector_Cluster->Servo_Timer_i32)->Value,
                  ServoTimer[MNum], SizeL);
            memcpy((*DPR_Real_Time_Vector_Cluster->Position_Dbl)->Value,
                  Position[MNum], SizeD);


            …
            // -- Indicate a valid buffer to caller and reset buffer counter

            BufferCount[MNum] = 0;
            *ValidData = LVTRUE;
      }
      else {
            *ValidData = LVFALSE;         // -- No valid buffer
      }

      PmacDPRSetHostBusyBit(DevNum, 0);        // PMAC can do its thing
      return noErr;
}

CIN MgErr CINLoad(RsrcFile rf)
{
      int i;
      // -- Reset the buffer counters
      for (i = 0; i < MOTOR_MAX; i++) {
            BufferCount[i] = 0;
      }
      return noErr;
}
```

It's a bit longer than **PmacDPRRealTimeMotor** but has much the same structure. The PMAC Busy Bit is processed as before. Instead of assigning the

fetched data directly to the items of the cluster, it is stored in a set of global arrays. The BufferCount array keeps track of the next array location to be written when valid data is available for storage. There is a cryptic **if** clause involving the ServoTimer data and BufferCount. The actual storage of fetched data in the temporary arrays requires that the ServoTimer not be the same as that of the last sample. Otherwise, the array would contain multiple samples of the data.

After the temporary arrays are updated the CIN check the current buffer count against the desired buffer size. If there are enough samples in the temporary array, the data will be copied from the temporary arrays into the cluster of arrays passed back to the VI. This requires you to resize each array in the cluster using LabVIEW's **NumericArrayResize** and then copy the data from the temporary array into the data buffer that will be returned to the VI. Once you get the basics of this requirement clusters and arrays are very easy to handle in your CIN.

There are some conditions associated with this approach. The temporary storage arrays have a fixed size defined by MOTOR_MAX and BUFFER_MAX. They have a native 2 dimensional C organization. As compiled you cannot ask for a Motor Number larger than MOTOR_MAX or gather more samples than BUFFER_MAX. The CIN does not check this condition. If you wish to resize these because of memory limitations or you want larger buffers you need to change these values, recompile the CIN, and reload it into the CIN in **PmacDPRRealTimeVectors**.

It is possible to allocate these buffers dynamically using various CIN utilities. However, this introduces more complexity to the process such as allocating the buffers in the function CINLoad and deleting the buffers in CINUnLoad. For this release of PMACPanel, this approach was not utilized.

A Note About Vector CINs

To avoid unnecessary complication we have not provided bullet proof **PmacDPR** VIs with error diagnosis and such. You should be aware of the fact PComm32 handles a lot of book keeping issues associated with DPR. As an example, the order in which you configure and enable DPR operations is important. If you enable a Variable Background buffer after you enable a Fixed Background buffer then disable the Fixed Background buffer the Variable Background buffer may move. Hence, those VIs accessing it will not return the correct data.

A similar issue arises when using the Vector VIs. Once you have enabled a particular Vector buffer for a specific number of samples DO NOT change the length. If you do you should unload the VIs that use the Vector VIs and reload them so that the buffer management can be reinitialized. Otherwise, it is highly probable that buffer bookkeeping will become garbled and strange things will happen.

A Note About Vector CIN Reentrancy

Many of the **PmacDPR** VIs that use CINs are reentrant. This is generally not a problem unless you are not careful how you use them. If you have two Vector VIs buffering servo data for the same motor you will get strange results. Sometimes one of the VIs will update the BufferCount and sometimes the other VI will update the count. Eventually, one of the two VIs completes the acquisition and gets the vector data leaving the other one without an acquisition. Generally this is a mistake in your application logic and can be remedied by handling the distribution of the acquired vectors in your diagram. One should use one Vector VI per motor thereby guaranteeing no need for mutexs to control access to the temporary data buffers.

PmacDPRFixedBack

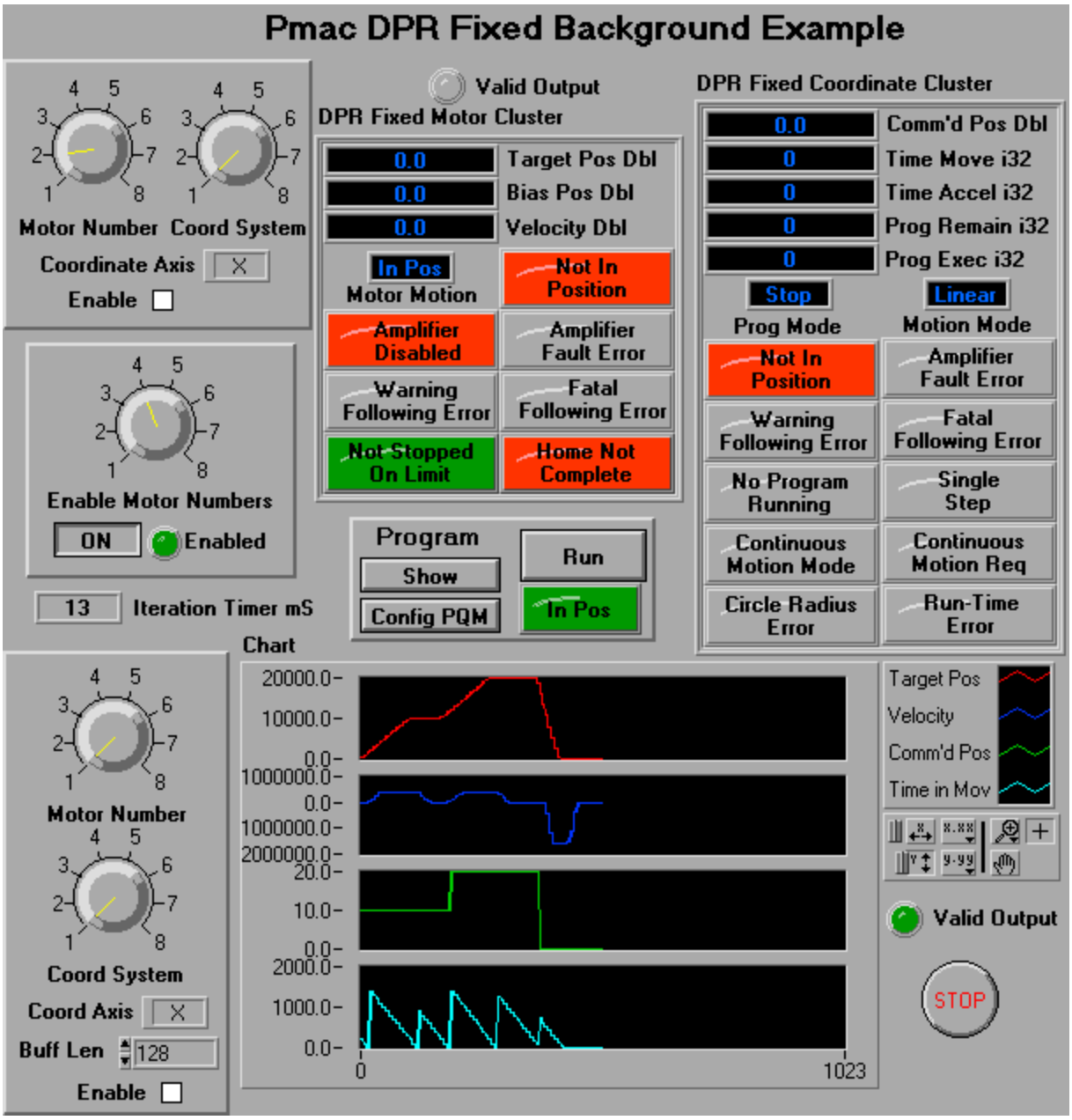
PMAC's Fixed Background data buffer mechanism automatically copies 34 selected Motor Calculation Registers, Coordinate System Control Registers, and Program Execution Registers from their native PMAC locations to DPR locations when requested. Whereas the Fixed Real Time data buffer is motor specific the Fixed Background data is Motor and CS specific, therefore, program specific. PComm32 supplies a set of routines to read these from DPR and convert them into legitimate Intel formats. This process hides the required handshaking between PMAC and the host to avoid collisions when accessing DPR. Update of a particular item is not synchronized with a specific servo cycle.

To minimize your work as a developer and simplify the interface the **PmacDPRFixedBack** collection of VIs has three VIs that collect the 28 most useful items and bundle them into LabVIEW clusters. The gathering of Fixed Background data is not controlled by the servo clock therefore the data items might indeed be taken at slightly different servo times.

PmacDPRFixedBackExample

The following example demonstrates all three **PmacDPRFixedBack** VIs. One to configure and enable the operation of Fixed Background data buffering, one to fetch the data for a specific Motor/CS, and one to buffer a set of vectors. **PmacDPRFixedBack** collects the data for a single motor operating in a CS. **PmacDPRFixedBackVectors** buffers the data exactly as **PmacDPRRealTimeVectors** does.

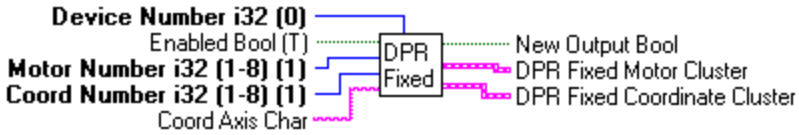
The panel for the example is shown below. The panel demonstrates the fetching of Fixed Background data for a single Motor/CS on top and the fetching of vectors for charting on the bottom. On the left are controls for selecting which motor and CS to use for the fetch and enabling the fetch from DPR. The example is different from **PmacDPRRealTimeExample** in that because the Fixed Background data buffer handles motor, CS, and program information it has an encapsulated motion program that can be configured and run using the box of buttons in the middle.



To execute the example select how many motors you want PMAC to copy to DPR using the knob labeled Enable Motor Numbers and click the Enable button. You can enable the operation of the display clusters by checking the box labeled Enable on the top left. There are two clusters provided by **PmacDPRFixedBack**. A DPR Fixed Motor Cluster for the specified motor and a DPR Fixed Coordinate Cluster for the specified CS. If you click the Run button, the data in the clusters will update.

- **PmacDPRFixedBack** - Once DPR Fixed Background buffer operation is enabled this VI can be used to fetch the data for a specific Motor Number and Coord Number. The input Enabled can be used to enable and disable the actual fetch. The Default, un-wired, condition is TRUE. Coord Axis Char is a string (X, Y, Z, A, B, C, U, V, W) indicating which axis in Coord Number Comm'd Pos will represent. When New Output is TRUE DPR

Fixed Motor Cluster and DPR Fixed Coordinate Cluster contain the most recent background data. When Enabled is FALSE the two output clusters contain the last valid data even though New Output is FALSE.

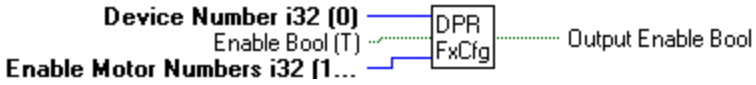


You will notice that there is a CS knob and a Coordinate Axis string. The values in the Coordinate cluster are for all motors in the CS while the Commanded Pos (Comm'd Pos) is for a specific Axis in the CS. In the example the 'X' axis is specified. If you change the axis to 'Y' and click Run again you will notice that the Comm'd Pos does NOT update because the 'Y' axis is not defined in CS 1.

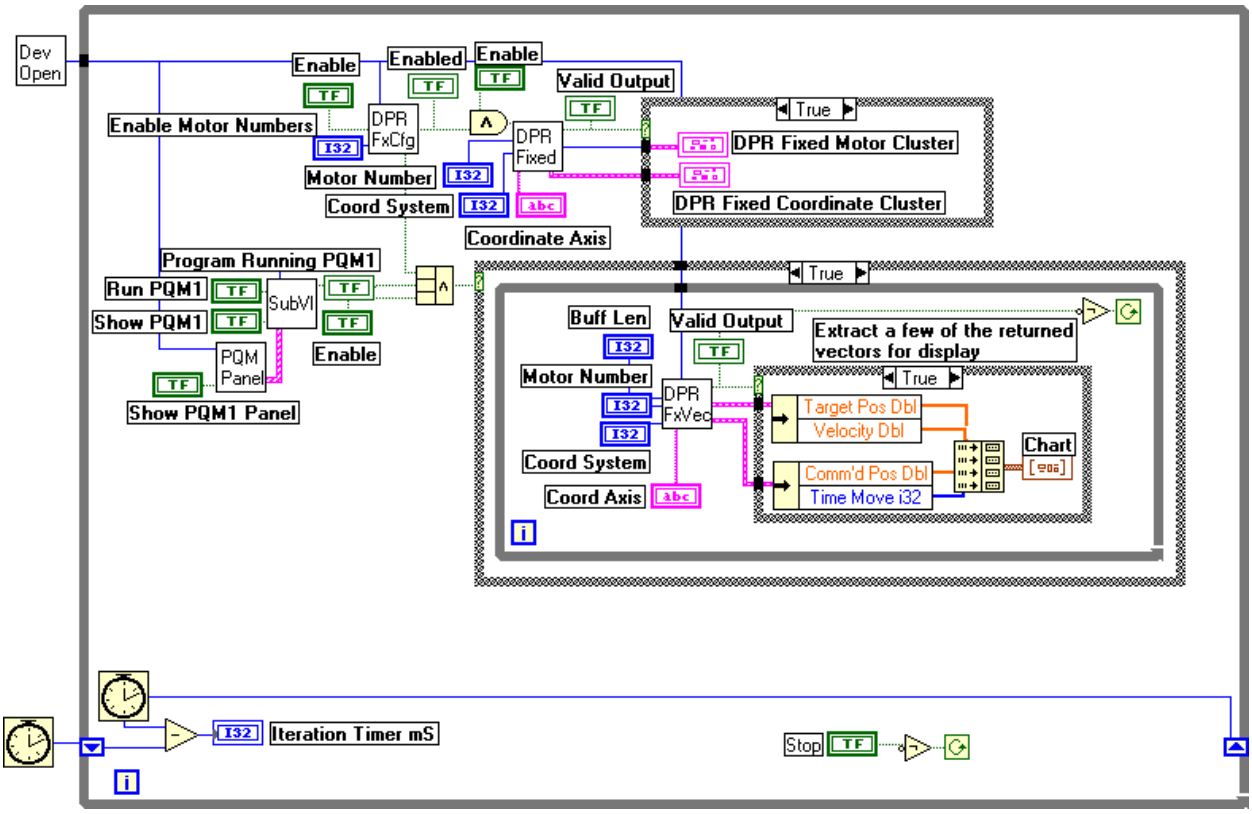
The operation of the example is a little different when you enable the Vector operation by checking its Enable box. Nothing happens until you run the program. This is due to the organization of the example not the associated Vector VI. When you enable the Vector portion of the example, and click Run the chart will begin updating. You should disable the top portion so that it doesn't steal samples from the Vector operation. Yo u will immediately notice that the displayed data is more quantized because of the DPR data is updated in the background. Fixed Background data is program related and therefore computed at a slower rate and updated only when requested. The other thing you should note is that the velocity for Fixed Background operation is in encoder counts per minute whereas Real Time motor velocity is in scaled Ix09 counts per servo cycle.

The diagram for the example shows three VIs. One to configure the operation, one to fetch the data for a single motor and one to fetch the vector data. In addition, there is a small diagram to handle the encapsulated motion program. A Case structure is used to control the update of the display clusters. The **PmacDPRFixedBackConfig** and **PmacDPRFixedBack** VI operate similarly to their **PmacDPRRealTime** versions in that the Enable terminals operate the same way.

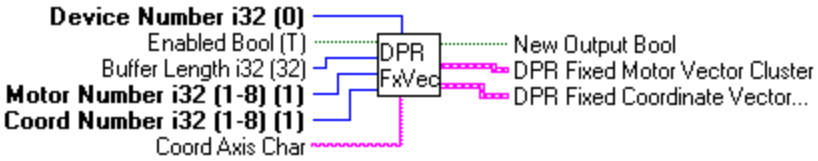
- **PmacDPRFixedBackConfig** - DPR Fixed Background buffer is configured to update motor information for all motor between 1 and Enable Motor Numbers when Enable is TRUE. Operation is disabled when Enable is FALSE. Output Enable is TRUE when operation is enabled. The state is maintained by the VI. Operation of DPR Fixed Background buffers overlaps with DPR Real Time operation in that the number of motors enabled must be the same.



You will note that the **PmacDPRFixedBack** vector VI is wrapped in a Case structure that is only executed when the configuration VI is enabled, the program is executing, and the proper Enable on the panel is checked. Hence, when the program is started the gathering of the vectors can begin. The samples are accumulated as fast as possible because the actual Vector VI is buried in a While structure that executes until the entire vector is accumulated and then the rest of the system gets a chance to run. This structure is not required but demonstrates another way to organize a gather.



- **PmacDPRFixedBackVectors** - Once DPR Fixed Background buffer operation is enabled this VI can be used to fetch the data for a specific Motor Number and Coord Number. The input Enabled can be used to enable and disable the actual fetch. The Default, un-wired, condition is TRUE. Coord Axis Char is a string (X, Y, Z, A, B, C, U, V, W) indicating which axis in Coord Number Comm'd Pos will represent. When New Output is TRUE DPR Fixed Motor Cluster and DPR Fixed Coordinate Cluster contain the most recent background data. When Enabled is FALSE the two output clusters contain the last valid data even though New Output is FALSE.



We will not discuss the details of the CINs associated with each VI. They are very similar to those presented for their cousins in the PmacDPRRealTime collection. Specific details are contained in source code comments.

PmacDPRNumeric

PMAC's architecture permits access to any unused DPR memory for whatever purposes you desire using M-Variables. These may be scratch registers or, in the case of DPR, registers through which you can pass data between the host and

PMAC. By avoiding the translation involved in standard ACSII communication you can move data between the host and PMAC much more quickly and it does not utilize precious P or Q Variables. The process requires no explicit handshaking between PMAC and the host to avoid collisions when accessing DPR.

The **PmacDPRNumeric** collection consists of a number of VIs that use Call Library Function VIs to access DPR. The simplicity of the interface doesn't require CINs. We show how to access individual memory locations as doubles, long (i32) and short (i16) integers, bit fields, and booleans. The examples use encapsulated PLCs and motion programs to generate and process the data transferred between the example PMACPanel applications and PMAC. If you use DPR for this purpose, the examples are very useful.

DPR Addresses and Data Organization

The mapping of memory addresses between the host computer on one side and PMAC's address space on the other side is simple. To PMAC, DPRAM appears as standard memory in the range $D000 - $DFFF, which can be thought of as 4K of long (48-bit) words or 8K of single (24-bit) X/Y words. This memory is accessed using M-Variables mapped to this address range. Depending on the DPR mechanisms used (Real Time, Fixed Background, Variable Background, etc.); the lower portions of this memory space are automatically allocated. Hence, you can use anything above the end of this space up to $DFFF for any purpose. There is no easy way to automatically allocate and addresses for DPR Numeric access until you have allocated all other automatic features. In the next section on DPR Variable Background buffers, we will tell you how to determine the end of this allocated space. For now assume that we will be working with DPR memory between &DE00 and &DFFF

To the host computer DPR appears as 8K 16-bit words of memory. Each 24 bit PMAC X or Y word thus takes two 16-bit memory addresses. A PMAC long or float (48-bits) thus takes four memory addresses. Fortunately, PComm32 handles the host computer memory mapping and PMAC handles the required bit and byte manipulations to map Motorola 56K data formats to Intel data formats.

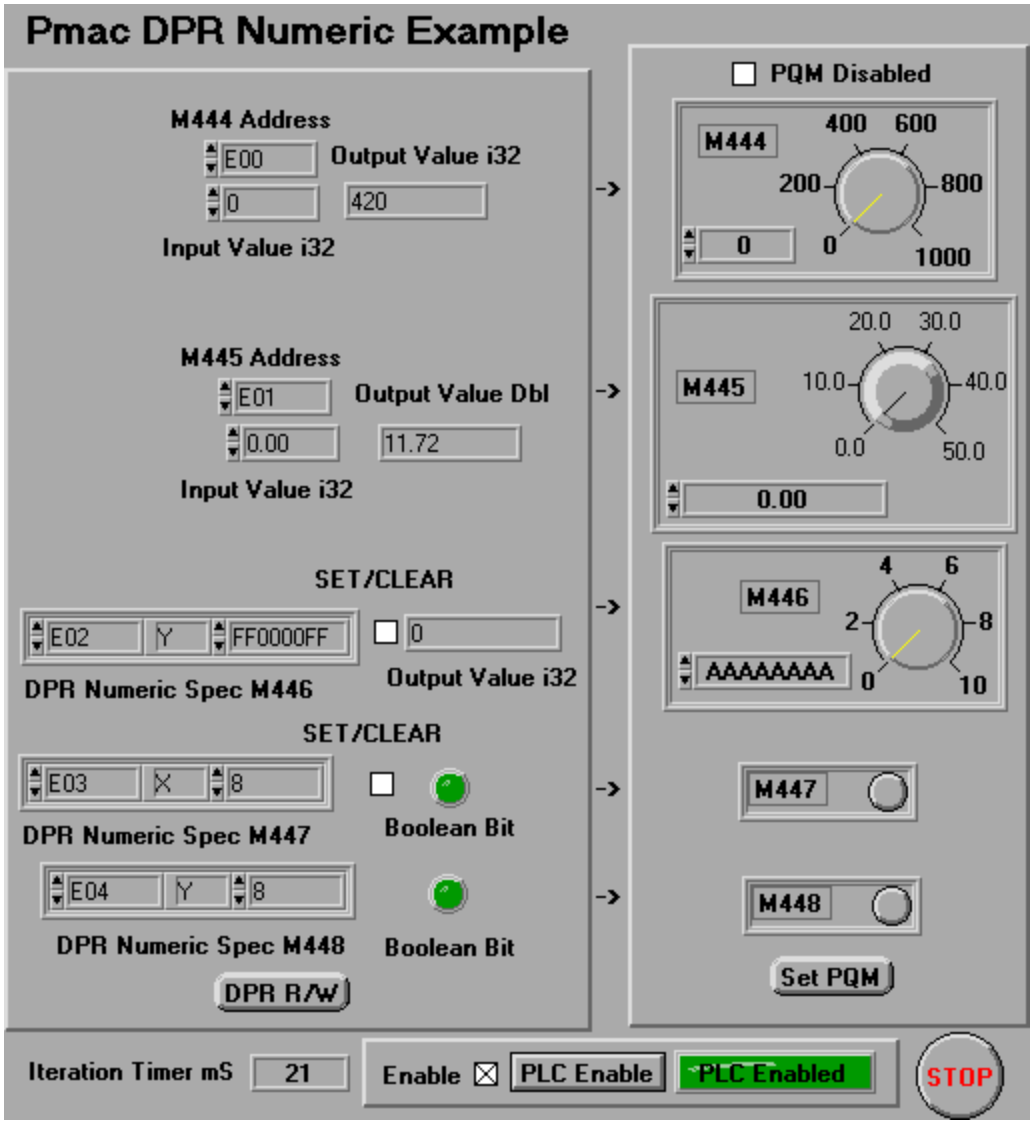
PmacDPRNumericExample

The following example demonstrates the use of DPR for communicating numeric data between the host and PMAC. PMAC executes a PLC that generates and responds to the register data accessed using M-Variables that are mapped to DPR. The data is also accessed using the PQM collection of VIs to demonstrate the differences in access speed, and bypass the mechanisms that field access introduces.

Unlike the previous **PmacDPR** examples that required a configuration step prior to accessing the data, **PmacDPRNumeric** requires no configuration other than assignment of memory addresses.

The panel for the example is shown below. On the left is a box containing controls/indicators that access M444 – M448 using **PmacDPRNumeric** VIs. On the right are several controls that access the same data using the ASCII PQM collection of VIs. Each of the M-Variables can be accessed using either method. From the top down are M444 an integer, M445 a double, M446 a bit field in a Y address, M447a single bit in an X address, and M448, another bit in a Y address. Access to the M-Variable data using DPR requires an address, bit number, or field specifier. These are also shown on the left portion of the panel.

At the very bottom are a few controls to enable and monitor an encapsulated PLC that generates M-Variable data that is simultaneously available to the host and PMAC because it exists in DPR. If you check the box labeled Enable then click the button labeled PLC Enable, the PLC program will begin executing. The indicators for each of the M-Variables will immediately begin updating with the data being generated in PMAC's PLC. If you check the box labeled PQM Disabled on the right these indicators will also begin updating with a noticeable increase in the interval timer. This is because of the large overhead required to process the required ASCII commands.



M-Variables and VI Address Specification

Before getting into the example deeper lets look at the PLC M-Variable definitions shown below. These specify the addresses where PMAC will place the data during its writes to M-Variables and fetch the data when it reads an M-Variable. The address modifier DP defines a 32 bit long integer in DPR handled as the lower 16 bits of both X and Y addresses. The F modifier defines a 32 bit

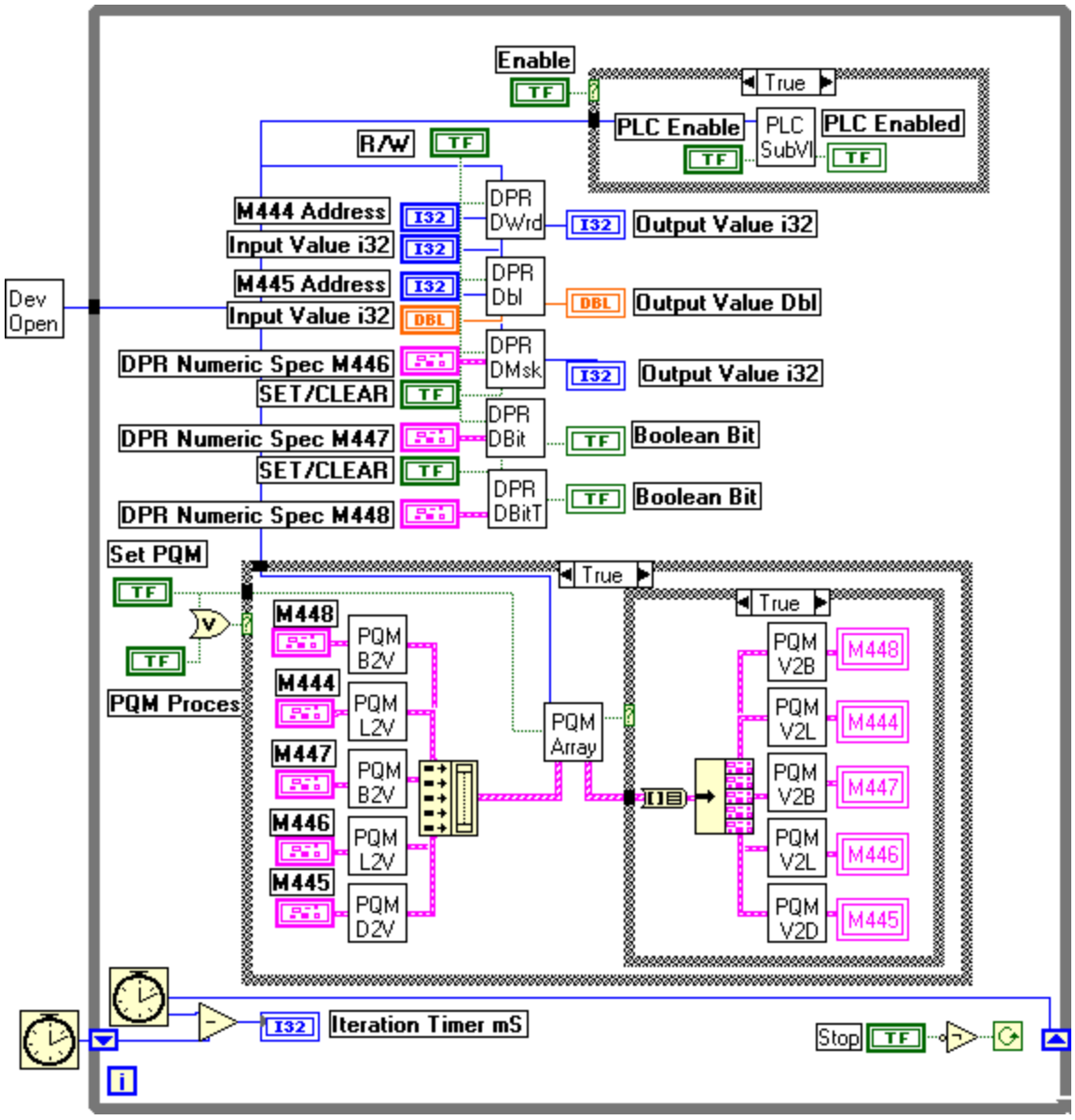
floating point value in DPR also handled as the lower 16 bits of both X and Y addresses. PMAC firmware and PComm32 handle the required bit and byte manipulations to convert the raw representation into Intel and Motorola formats. M447 and M448 are single bits defined in simple 24-bit X/Y words.

```
M444->DP:$DE00
M445->F:$DE01
M446->DP:$DE02
M447->X:$DE03,8,1
M448->Y:$DE04,8,1
```

To access these variables with the **PmacDPRNumeric** collection of VIs a truncated version of the memory address is required. A PMAC M-Variable defined at $DE45 become 0xE45 to PMACPanel. PComm32 handles the absolute memory mapping while the **PmacDPRNumeric** VIs compute the address offset required by PComm32. For M-Variables defined as F and DP, nothing more is required. For M-Variables defined as X, Y, or specific bit fields, a cluster defining the base address, modifier, and field or bit number is required. When looking at the panel only the address is required for M444 and M445. M446, M447, and M448 require the cluster. These are covered in detail later.

The diagram for the example has a section at the bottom for handling the PQM controls and a case statement at the top for handling the encapsulated PLC. The PLC encapsulation VI is wrapped inside the case statement so that it can be disabled and the impact of its execution on timing can be seen. The PQM approach can be used to validate the results of bit field manipulation that are masked by the DPR mechanism.

The five VIs in the middle handle the transfer of data between the host and PMAC using the same read/write architecture used for **PmacIVar**, **PmacMemory**, etc. All that is required is the address for DWord or Double memory or a DPR Numeric Spec cluster for field and bit access.



PmacDPRNumeric VIs are discussed in two groups. The first group covers word, double word, and floating point DPR M-Variables. The second group covers the bit and bit-field VIs. In a later example, we will demonstrate an approach for grouping collections of DPR M-Variables into a cluster that can be handled using a CIN.

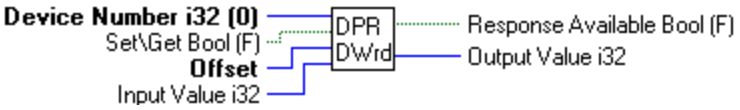
- **PmacDPRNumericDWord** - This VI is used to Set or Get PMAC long M-Variable's defined in DPR as M447->DP:$DE03. PMAC handles the translation of PMAC's representation into Intel format when the DP specification is used.

  When Set/Get is FALSE - default state - Response Available is TRUE and Output Value is the value in DPR located at Offset. When Set/Get is TRUE Output Value = Input Value, Response Available is FALSE and the specified DPR location is set.

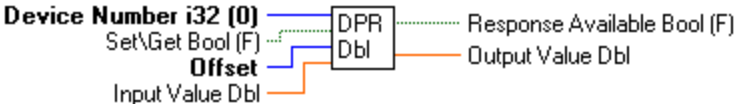
The mapping of PMAC addresses to PC addresses is involved and requires a bit of work to understand. In general, each 24-bit PMAC word requires one 32-bit PC word. For PMAC longs specified as.

M447->DP:$DE03

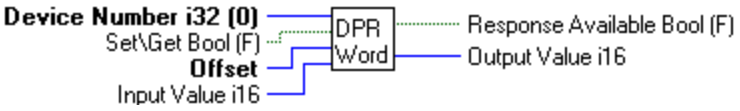
this offset should be $E03.



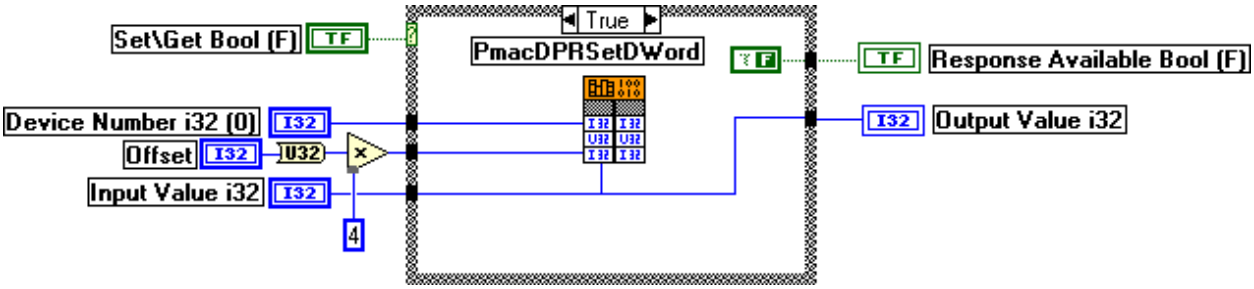
- **PmacDPRNumericDbl** - This VI is used to Set or Get PMAC double M-Variable's defined in DPR as M447->F:$DE03. PMAC handles the translation of PMAC's representation into Intel format when the F specification is used.



- **PmacDPRNumericWord** - This VI is used to Set or Get PMAC long M-Variable's defined in DPR as M447->DP:$DE03 where the equivalent intel representation is 16 bits. PMAC handles the translation of PMAC's representation into Intel format when the DP specification is used.



The implementation of these VIs closely follows that used by **PmacMemory** and **PmacIVar**. The VI will Get the specified value unless the Set\Get input is TRUE in which case it does a set operation. The diagram shown below demonstrates how this is done using a Call Library VI using the **PmacDPRSetDWord** function in PComm32. The FALSE case (Get operation) uses the **PmacDPRGetDWord** function. Note that the offset supplied by your diagram is multiplied by 4 to get the actual memory offset of the M-Variable in DPR as seen from the host.



DPR Bits and Bit Fields

The three VIs presented above only require the offset to determine the address of the desired data. When accessing bits and bit fields the information contained in the **PmacDPRNumericSpec** cluster is required.

DPR Numeric Spec Cluster A cluster of items required to describe a DPR mapped PMAC M-Variable for bit and field access.

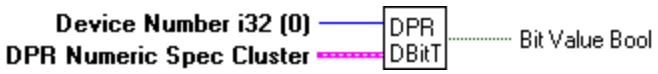
Address i32 Hexadecimal integer specifying DPR address offset. For example, PMAC Addresses such as:
M445->F:$DE01
M446->DP:$DE02
M447->X:$DE03,8,1
Become
E01, E02, and E03 respectively.M446->DP:$DE02

X/Y String A single character string (X or Y) defining the type of data. Not for L or DP.

Mask/Bit i32 A hexadecimal value used to define a bit number for single bit operations or a multi digit hexadecimal number defining a mask for multi-bit operations.

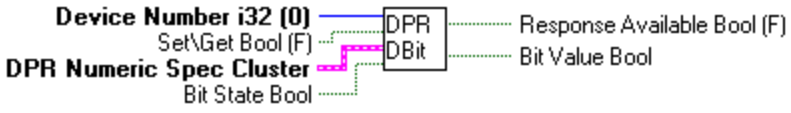
The VI **PmacDPRNumericSpec** is embedded in the bit and bit field VIs covered next, and converts the address specification into an actual DPR memory offset. You can look at the actual diagram for this VI if you wish to understand how this is done. Generally, although these are named as DWord operators the individual bits are defined in a 24-bit X/Y word. The bit VIs are

- **PmacDPRNumericDWordBitTest** - This VI queries the DPR DWord bit specified by DPR Numeric Spec Cluster and returns the value in Bit Value.



- **PmacDPRNumericDWordBit** - This VI operates on the DPR DWord bit specified by DPR Numeric Spec Cluster.

  When Set/Get is FALSE - the default state - the value of the bit is queried and returned by Bit Value with Response Available TRUE. When Set/Get is TRUE the specified bit is set to the value of Bit State - either TRUE or FALSE.

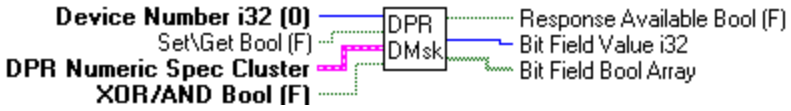


Bit field operations are a little more complex. The following VI allows you to specify an entire X/Y word and set or clear multiple bits in a single operation depending on the control input.

- **PmacDPRNumericDWordSetMask** - This VI operates on the DPR DWord bit field specified by DPR Numeric Spec Cluster.

  When Set/Get is FALSE - the default state - the Mask specified by DPR Numeric Spec Cluster is AND'd with the specified address to produce the output Bit Field Value. Response Available is TRUE.

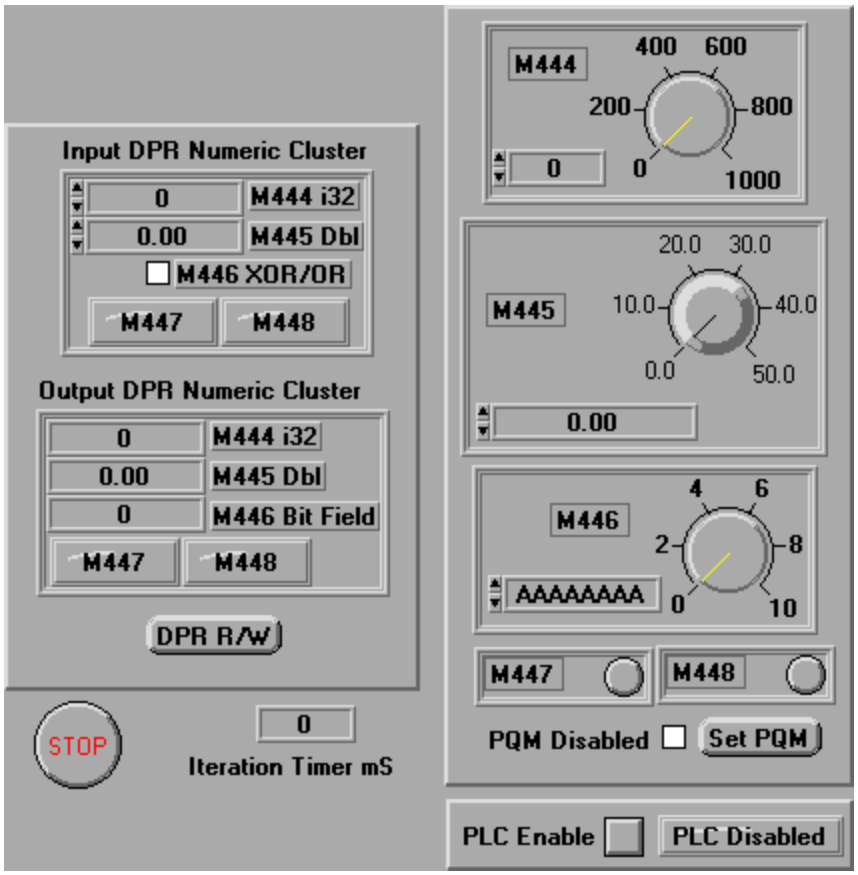
  When Set/Get is TRUE the Mask is either OR'd or XOR'd with the contents of the field at the specified address. If XOR/OR is FALSE the mask is OR'd with the contents of the field at the specified address thereby setting bits specified by the mask. If XOR/OR is TRUE the mask is XOR'd with the contents of the field at the specified address thereby clearing the bits specified in the mask.



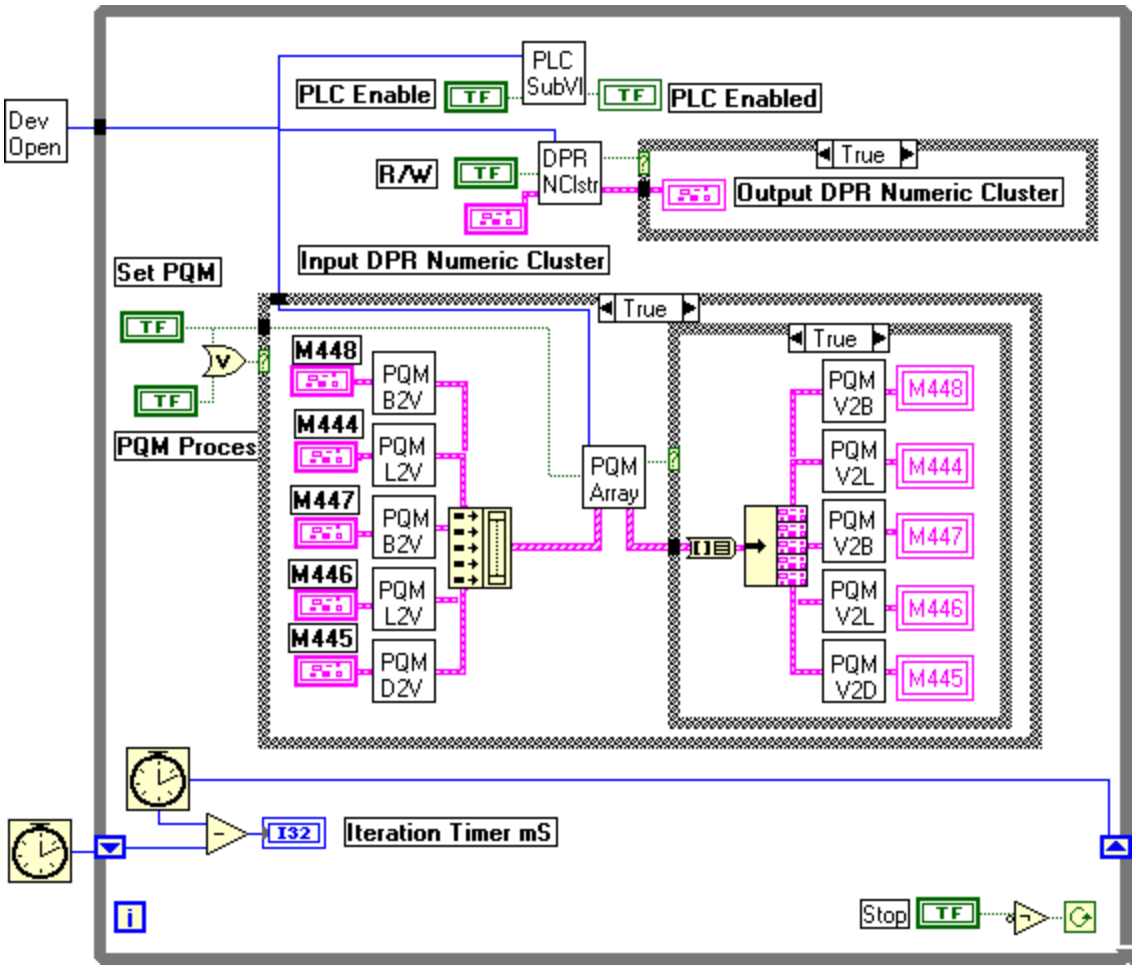
PmacDPRNumericClusterExample

This example extends the previous example by defining a cluster containing a set of DPR numeric data. The purpose of doing this is to hide the addresses inside the VI and get ready for another example that will use a CIN to access the DPR numeric data.

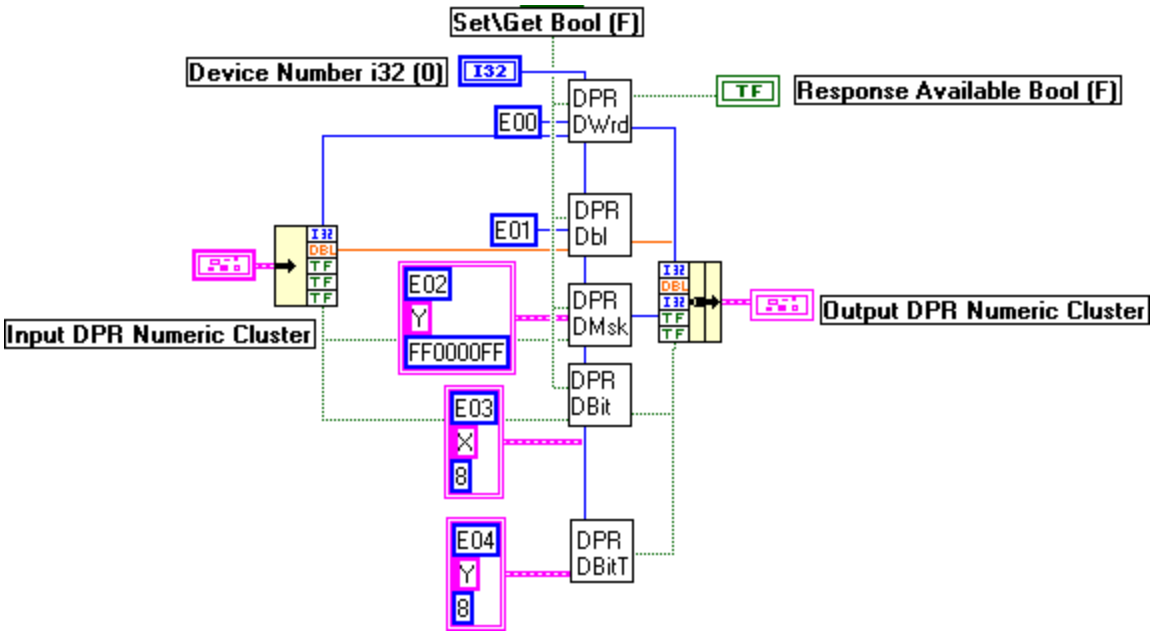
The example operates the same as the previous one except that a large number of controls for defining the M-Variables have been reduced to a single cluster as shown in the panel.



The handling of the five VIs in the earlier example diagram is reduced to a single VI.



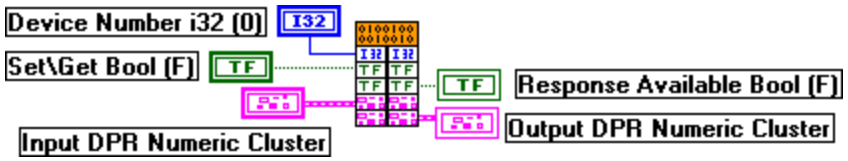
The diagram for the DPR Numeric Cluster VI is shown below. You can copy this and modify it to support your own requirements. Note that the addresses are specified by cluster constants. These can be created using the right mouse button and selecting **Create Constant**.



PmacDPRNumericCINClusterExample

This example takes the previous example a little further and implements the actual handling of DPR data using a CIN. This can be useful if you have many data being transferred or have trouble maintaining dozens of **PmacDPRNumeric** VIs or have special data handling requirements that benefit from a CIN.

The diagram handling the 5 M-Variables in the example above are reduced to a single CIN VI that will handle the reading and writing of the data with direct PComm32 calls. To simplify development of these CINs, **PmacDPR** defines a set of macros that make life very easy.



The code for the CIN is shown here. To use the macros include the file **PmacDPRNumericCINCluster.h** located in **PmacDPR**.

```
#include "extcode.h"

// -- PmacDPRNumeric Macros --

#include “PmacDPRNumericCINCluster.h"
#include <pmacu.h>
#include <dprrealt.h>

/* typedefs */
```

```
typedef struct {
      int32 M444_i32;
      float64 M445_Dbl;
      int32 M446_Mask_i32;
      LVBoolean M446_XOR_OR;
      LVBoolean M447_Bit;
      LVBoolean M448_Bit;
} TD1;

typedef struct {
      int32 M444_i32;
      float64 M445_Dbl;
      int32 M446_Bit_Field;
      LVBoolean M447_Bit;
      LVBoolean M448_Bit;
} TD2;

CIN MgErr CINRun(int32 *Device_Number_i32_0_, LVBoolean
                  *Set_Get_Bool_F_,
                  LVBoolean *Response_Available_Bool_F_,
                  TD1 *Input_DPR_Numeric_Cluster,
                  TD2 *Output_DPR_Numeric_Cluster) {

      int32 DevNum = *Device_Number_i32_0_;

      // -- Using the macros

      PmacDPRNumericDWord (*Set_Get_Bool_F_,
                  0xE00,
                  Input_DPR_Numeric_Cluster->M444_i32,
                  Output_DPR_Numeric_Cluster->M444_i32);

      PmacDPRNumericDouble(*Set_Get_Bool_F_,
                  0xE01,
                  Input_DPR_Numeric_Cluster->M445_Dbl,
                  Output_DPR_Numeric_Cluster->M445_Dbl);

      PmacDPRNumericDWordMask(*Set_Get_Bool_F_,
                  0xE02,
                  Input_DPR_Numeric_Cluster->M446_Mask_i32,
                  Input_DPR_Numeric_Cluster->M446_XOR_OR,
                  Output_DPR_Numeric_Cluster->M446_Bit_Field);

      PmacDPRNumericDWordBit (*Set_Get_Bool_F_,
                  0xE03, 'X', 8,
                  Input_DPR_Numeric_Cluster->M447_Bit,
                  Output_DPR_Numeric_Cluster->M447_Bit);

      PmacDPRNumericDWordBitTest (0xE04, 'Y', 8,
                  Output_DPR_Numeric_Cluster->M448_Bit);
      PmacDPRNumericResponse(*Response_Available_Bool_F_,
                              *Set_Get_Bool_F_);
      // --

      return noErr;
}
```

As with all CIN nodes, LabVIEW writes the function declaration and defines the parameter types. The macros require the device number be defined DevNum whatever it is in your parameter list. The macros perform a conditional test for read/write operations, address calculations, and PComm32 operations. Generally the macros require an address and a pointer to the input and output elements of the cluster. Bit operations require the bit number and Mask or Bit Field operations require the mask, field mask, and XOR/OR operator. The actual C-code for the example contains the macros and their actual C-counterparts to illustrate the operations performed. If you've followed the examples so far, the operation of the macros will be obvious.

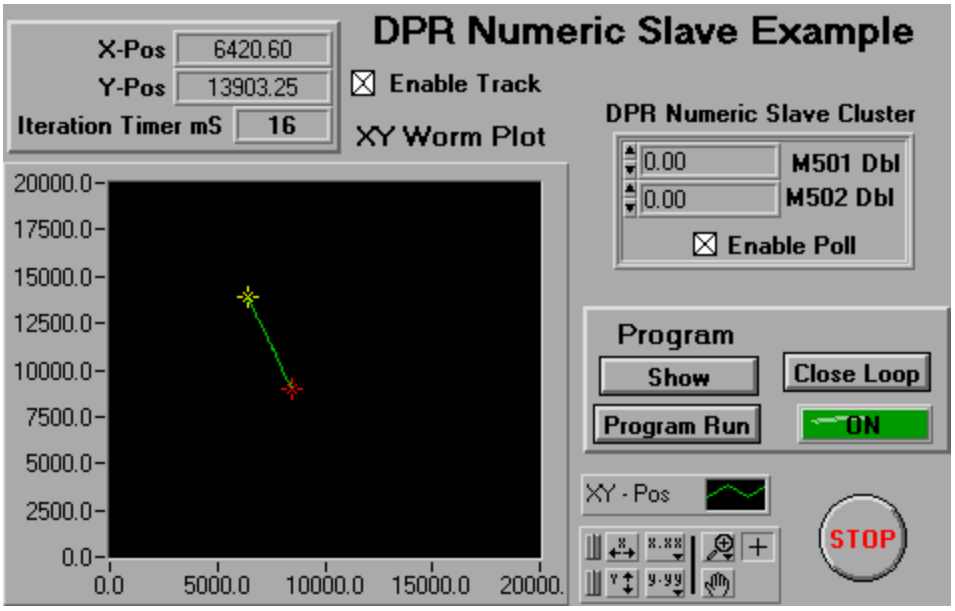
PmacDPRNumericSlaveExample

This example uses **PmacDPRNumeric** and **PmacDPRRealTime** capabilities to build an application that allows the user to move a 2-axis X-Y table with the mouse. To accomplish this the motion program shown below uses DPR mapped M501 and M502 to define the target position for motors 3 and 4. M500 is a Boolean used to control a loop that breaks the target position into a set of smaller moves. The program is encapsulated in a VI for easy use.

```
; USE CS &3

&3
#3->10x
#4->10y

; -- These are DPRAM mapped target coordinates

M501->F:$DE06    ; X coordinate
M502->F:$DE07    ; Y coordinate

; -- RUN BOOL

M500->Y:$DE05,8,1

; --

open prog 61 clear
;hm1..4
i13 = 100

P209 = 1000    ; Vector distance per increment
P211 = 10000    ; Starting positions

M501 = 10000
P212 = 10000
M502 = 10000

tm10

; -- Move tracking

while (M500=1)

    P221 = M501 - P211    ; Cartesian distance to go
    P222 = M502 - P212
```

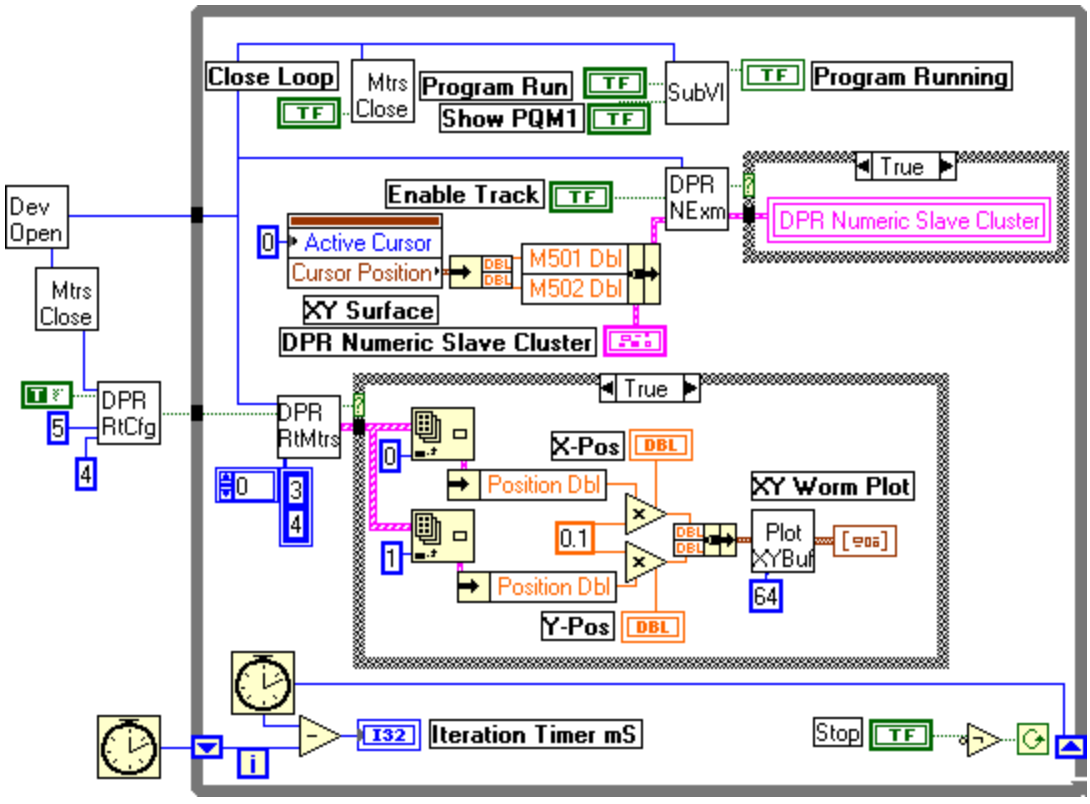
```
    P200 = sqrt(P221 * P221 + P222 * P222)
    if (P200 > P209)      ; If longer than increment
        P200 = P209 / P200    ; Fraction of distance
        P211 = P211 + P221 * P200
        P212 = P212 + P222 * P200
    else
        P211 = M501
        P212 = M502
    endif
    x(P211)  y(P212)
endwhile
close
```

The panel for the example is shown below. When the VI is running, and Enable Track is checked, the values in DPR Numeric Slave Cluster - M500, M501, and M502 - are continuously written to memory for use by the motion program. Checking Enable Poll sets M500 TRUE so that clicking the Program Run button enables the program loop. If you don't check the box prior to starting the motion program, the loop in the motion program will not execute. The Close Loop button is provided to close the servo loops if they are not closed or the initial rapid move to home results in a fatal following error.



The indicators on the middle left of the panel display the actual X/Y motor positions as retrieved by **PmacDPRRealTimeMotors**. The Yellow cursor in the plot is the target position for the move and determines the values of M501 and M502. When you click the Yellow cursor and drag it to a new position the target position for the move is set and executed by the motion program. The Green worm will begin moving toward the Yellow cursor with the Red cursor bringing up the rear.

The diagram for the example is similar to those already discussed. The motion program wrapper VI on top handles the program execution. Below this is the VI handling the DPR Numeric cluster containing M500-M502.



The query for motor positions used to update the plot is provided by **PmacDPRRealTimeMotors**. The configuration of this capability is done outside the execution loop. When the motor positions are fetched, array is indexed, the position are unbundled, converted to CS units (multiplied by 0.1), used to update the panel indicators, and bundled into and X-Y point for the X-Y Chart buffer. Remember that **PmacDPRRealTimeMotors** does not perform CS conversions. We can also use the **PmacDPRFixedBack** VIs to obtain the motor position in CS units.

Target position generation for the move is handled by retrieving the position of Cursor 0 from a chart attribute node, and updating the values in the DPR Numeric Slave Cluster. These values will be written by **PmacDPRNumericSlaveCluster** as long as Enable Track is TRUE.

PmacDPRVarBack

PMAC's Variable Background data buffer mechanism allows you to specify 128 memory addresses to copy from their native PMAC locations to DPR locations when requested. Whereas the Fixed Background data buffer allows access to predefined registers and locations, the Variable Background mechanism allows to you to access anything. PComm32 supplies a set of routines to read the copied data from DPR and convert them into legitimate Intel formats. This process hides the required handshaking between PMAC and the host to avoid collisions when accessing DPR. PComm32 supports a limited ability to write to DPR from the host and copy this data to its native location.
PmacDPRVarBack provides hooks for this interface but does not currently implement this capability.

To minimize your work as a developer and simplify the interface the **PmacDPRVarBack** collection of VIs provides three VIs. One to configure one or more Variable Background buffers, one to fetch its contents, and one to buffer the data into vectors. To aid you in specifying items the configuration VI uses the PmacGatherSpec cluster that forms the heart of the **PmacAddress** and **PmacGather** collections.

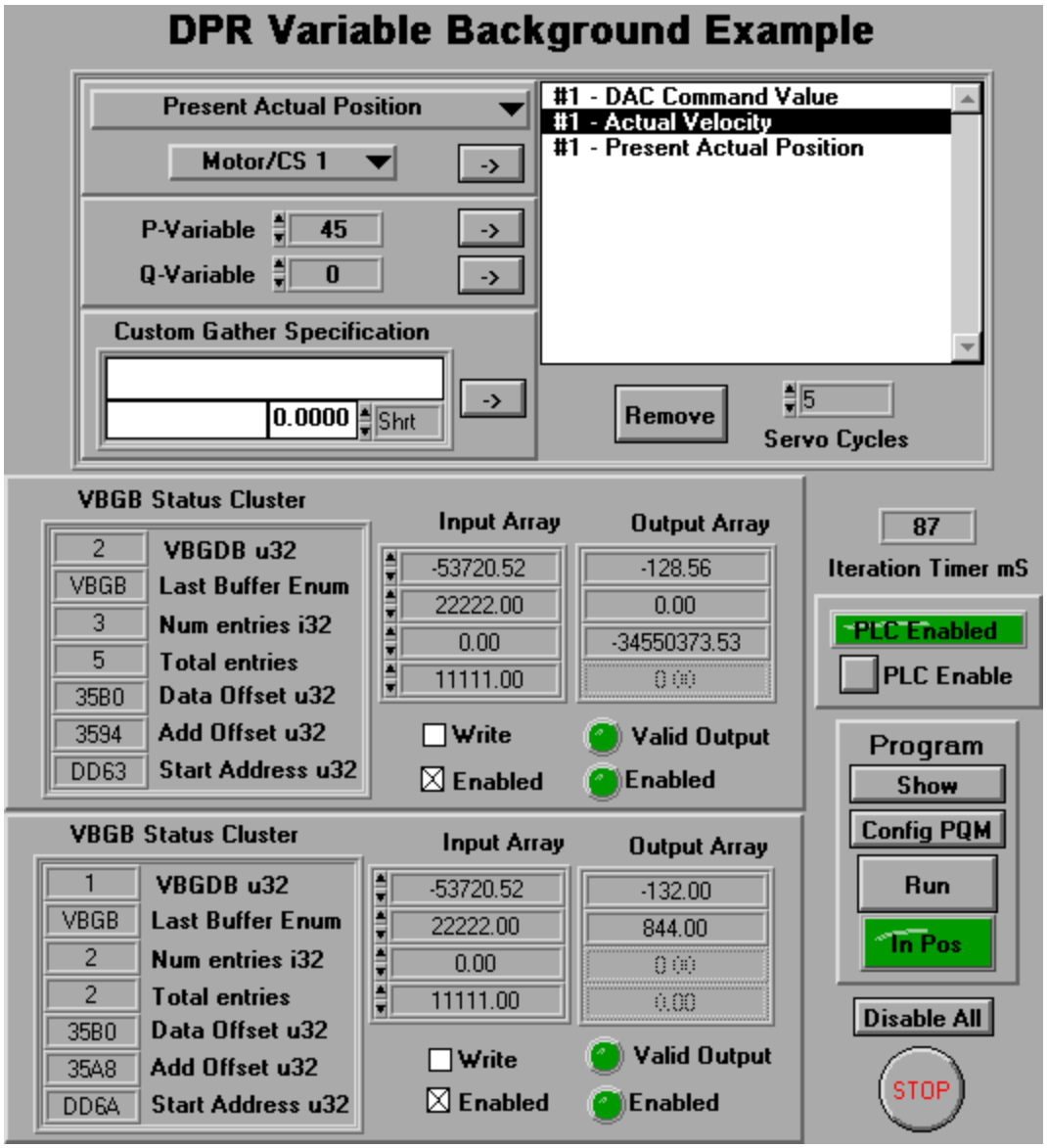
The gathering of Variable Background data is not controlled by the servo clock therefore the data items might be taken at slightly different servo times.

PMAC's Variable Background mechanism is very sophisticated and an integral part of PMAC. We will cover some specific issues you will encounter and must be aware of when using it.

PmacDPRVarBackExample

The following example demonstrates the configuration of two Variable Background data buffers. This allows you to build buffers that support your specific requirements. For instance, you can declare one buffer for each motor and coordinate system in your system. You can then declare another one for each I/O device, and one to monitor a collection of miscellaneous items. You can gather some of them as vectors, some for indicator clusters, and some for background computations. The only limitation is that you don't declare more than 128 items between all of them.

The panel for the example, shown below, allows you to define two independent buffers. The support clusters and such are collected into boxes in the bottom left quadrant. Each buffer has a VBGB Status Cluster containing information about the individual buffer, it location in DPR, and the entire pool of buffers. To the right of this are an Input Array and an Output Array. The Output Array contains the data for the specified buffer. The Input Array is provided but, as noted, not supported. Below these items is a Write check box (not supported), an Enabled check box, and to indicators. On the far right are a few collections of buttons to control the associated PLC and motion program. You have encountered these several times.



To run the example you need to specify a set of items to gather. This is done using the **PmacGatherSelect** cluster on the top left. The operation of this is detailed in the section on **PmacTerminalGather**. As shown in the example, there are three items defined. To create a Variable Background buffer, select the items you want in the buffer and check an Enabled box. In the example, the upper set was created to handle these three items. The lower set was created to handle P44 and P45 that are generated by the PLC program.

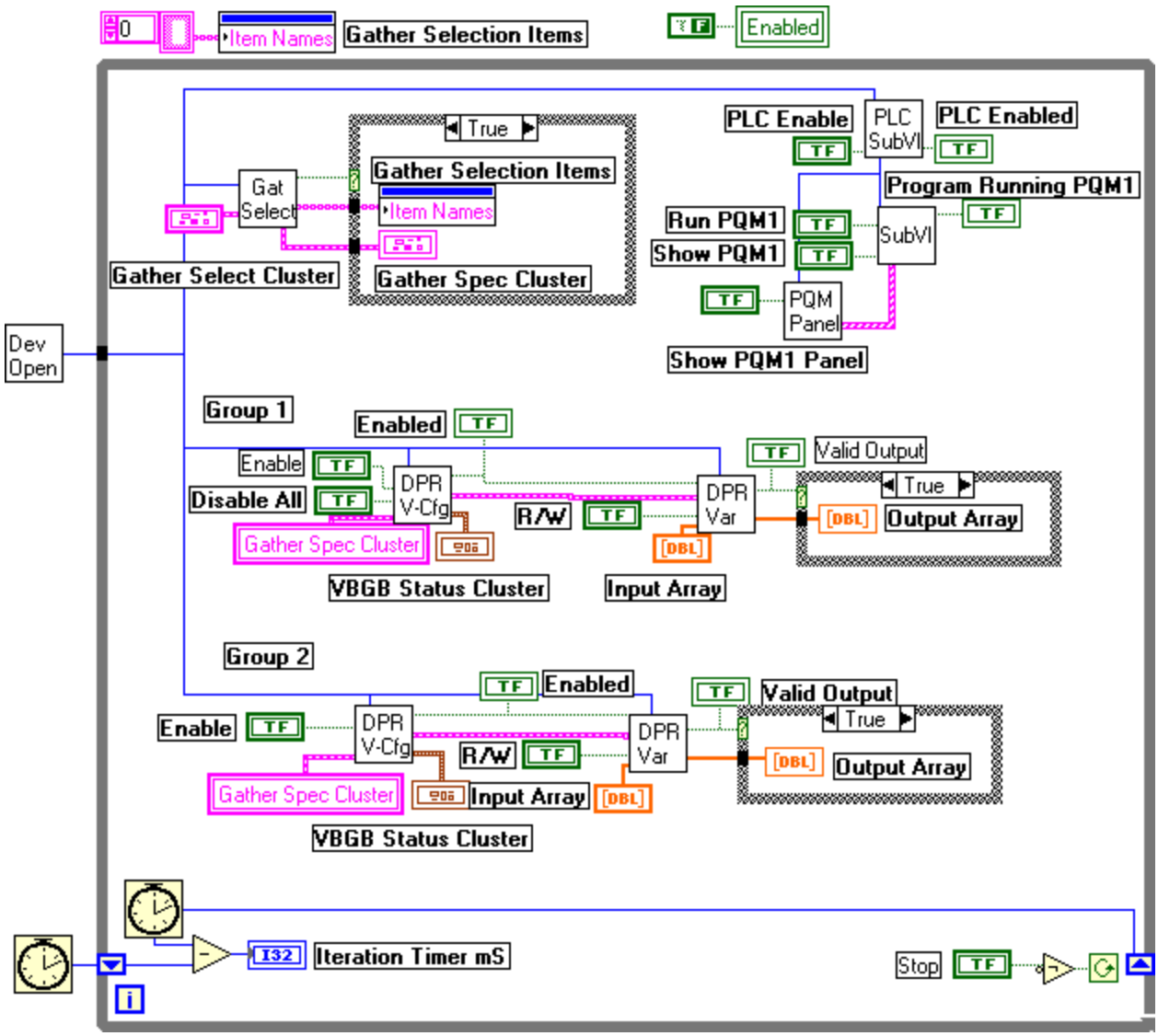
The VBGB Status Cluster maintains information about the individual buffers and the entire pool. Refer the VI Reference for details of each item. Here we will discuss what these clusters show. The top status cluster item VBGDB indicates that it is buffer 2 whereas the bottom cluster indicates it is buffer 1. Buffers are assigned in the order created. Num Entries indicates that the top buffer has three items (as defined by the Gather Spec) and the bottom buffer has 2 items. When the first buffer was created the Total number of entries was 2. After the second buffer is created, the Total number of entries is 5. Start Address indicates the start of the buffer in DPR. This is the last location

reserved by DPR. The Start Address of the first buffer created indicates the first address you can use for DPR Numeric access.

Notes on the use of PmacDPRVarBack

Variable Background buffers should be created AFTER all other DPR mechanisms have been enabled. PComm32 might and sometimes does moves things around when you start reconfiguring DPR. If you create more than one buffer DO NOT delete a previously created buffer. Again, PComm32 will shift things around and it is very likely that your remaining buffers will contain garbage. If you do delete a buffer you should delete and recreate the remaining buffers. You are encouraged to try this using the example. If you un-check the Enabled box that buffer is deleted. Chances are VERY HIGH that the remaining buffer will give you garbage data. Un-check the remaining buffers and then re-check them. Things will now behave as expected.

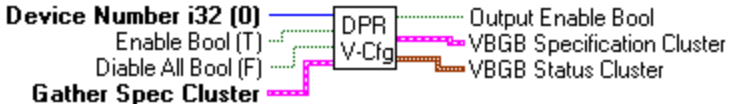
The diagram for this example demonstrates how easy it is to create and access a buffer. At the top left is the VI to handle the Gather Select Cluster. Check the section on PmacTerminalGather if you have questions. On the top right are the PLC and motion program handlers.



There are two almost identical configurations to handle the two buffers. Each consists of a configuration VI **PmacDPRVarBackConfig** and **PmacDPRVarBack** to actually fetch the data. The configuration VI requires a Gather Spec Cluster and produces a VBGB Specification Cluster for the handler. The VBGB Status Cluster is not required by other VIs but serves a useful diagnostic purpose.

- **PmacDPRVarBackConfig** - This VI creates a set of Address Items specified by Gather Spec Cluster using the DPR Variable Background when Enable is TRUE. The VI produces a VBGB Status Cluster with relevant information about this buffer and a VBGB Specification Cluster containing information required to actually fetch the data using PmacDPRVarBack and PmacDPRVarBackVectors.

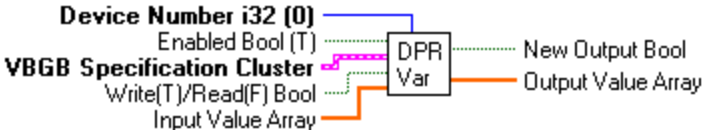
  Operation is disabled when Enable is FALSE. Output Enable is TRUE when operation is enabled. The state is maintained by the VI. This VI can be used multiple times to create sets of VBGB Address Items. See the documentation for limitations on how many sets can be created and their size.



Variable Background buffers allow you to gather the contents of any memory location, X, Y, DP, etc. To handle all data types the data fetched from DPR is all treated as a double. This allows **PmacDPRVarBack** to treat the fetched items as an array rather than have VIs to handle each type or implement a complex typing mechanism. If you need to access bits from a particular item, index the array, convert it to an integer, and use it as you wish. The Input Array and R/W inputs are not supported yet.

- **PmacDPRVarBack** - DPR Variable Background buffer operation is enabled this VI can be used to fetch the data specified during the configuration. The input Enabled can be used to enable and disable the actual fetch. The Default, un-wired, condition is TRUE. When New Output is TRUE Output Value Array contains the most recent background data. When Enabled is FALSE Output Value Array contains the last valid data even though New Output is FALSE.

  The Write/Read and Input Value Array inputs are not currently functional. Future releases may implement this capability.

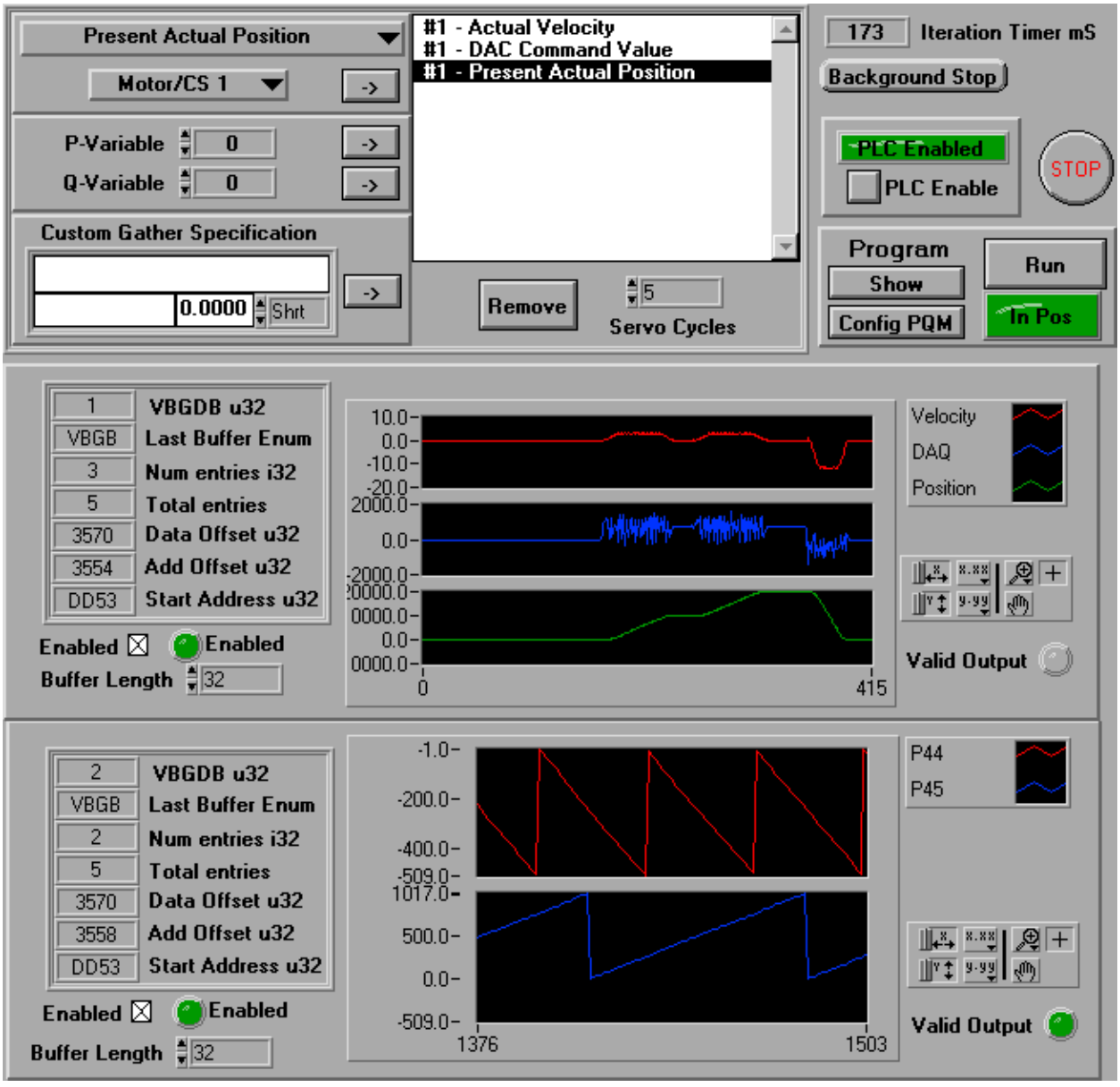


Note on Supporting PmacDPRVarBack CINs

The CINs to support this collection of VIs are significantly more involved than the previous collections. We will not cover these in the manual. If you understand those presented earlier and understand the information covered here, you can examine the C-code for yourself. The comments provide enough information if you desire to tackle changes yourself.

PmacDPRVarBackVectorExample

This example replaces **PmacDPRVarBack** with **PmacDPRVarBackVectors**. It use is almost identical to **PmacDPRVarBackExample**. The only difference is that the bottom buffer has a **PmacGatherSpec** cluster defined as constant containing entries for P44 and P45. Therefore, you do not have to specify the Address Items. Simply check its Enabled box. The diagram is not presented for the same reasons.



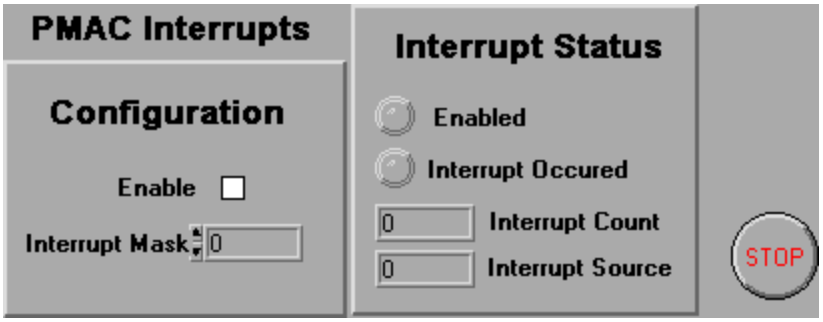
Chapter 12 - Interrupts

Basics

This Chapter documents an emerging interface to PMAC's interrupt system. The information presented here is preliminary and not supported yet.

PmacInterruptExamp

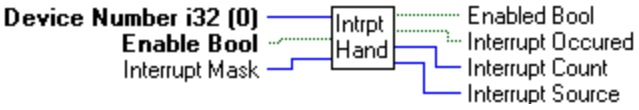
To run this example check the Enable box. Interrupt Mask defines which sources are enabled. The default value of zero enables all sources. The In Position flag generates an interrupt and is a good test. When an interrupt occurs the Interrupt Occurred LED is on and the count and source update.



ALWAYS disable interrupts when you application halts.

- **PmacInterruptConfig** - When Enable is TRUE enable interrupts. Indicate the availability of a handler by the output Enabled. When FALSE disable interrupts. ALWAYS disable interrupts when you application is not executing.

  Interrupt Occurred is TRUE whenever this VI checks the handler and determines an interrupt has occurred. Interrupt Count indicates the number of interrupts since the last service. Interrupt Source specifies which source. See *PMAC User Manual*s for details.


Device Number i32 (0)
Enable Bool
Interrupt Mask
Intrpt
Hand
Enabled Bool
Interrupt Occured
Interrupt Count
Interrupt Source

Glossary of Terms

Index

•

Position Capture Control "Encoder I-Variable 2" for Encoder n (I902; I907 · 153
Flag Select Control "Encoder I-Variable 3" for Encoder n (I903; I908 · 153

A

Addressing
  Coordinate Systems · 52
  Motors · 52
Applications
  Sample · 32
Axis Definition Statements · 152, 153

B

Buffer length · 44
  limitations · 47

C

Clusters
  Limits and types · 59
Code Interface Nodes · 32
Communication Buffers
  Maximum length · 20
Compare Control Bits · 173
Connecting PMAC
  Digital Inputs and Outputs · 184
CONTROL-F Command · 14, 16, 26, 43, 44, 52, 55, 72

D

Data gathering · 32
Development Tools · 31
Device configuration
  Special options for serial · 43
Digital Inputs and Outputs
  Input Source/Sink Control · 185
  Option for Sourcing Outputs · 185
  Software Access · 185
  Standard Sinking Outputs · 185
Display
  Resolution · 10
DPR
  Data Buffers · 28
  Requirements · 9
  VIs to access · 32

E

Encoder/Flag I-variables · 153
Encoders · 30
Error
  Automatic handling · 46
  Dialog box · 46

H

Help facilities
  LabVIEW On-line help · 16
Home Command · 153
Home Flag · 153
Home Speed for Motor x (Ix23) · 153
Homing · 30
Homing from a PLC Program · 155
Homing Into a Limit Switch · 155
Homing Search Move · 152, 153
  Action on Trigger · 153
  Home Command · 153
  Homing from a PLC Program · 155
  Homing Into a Limit Switch · 155
  Storing the Home Position · 154
  Zero-Move Homing · 155
Homing Speed · 153

I

Input/Output
  Compare-Equals Outputs · 183
Installation
  Configuring PMACPanel · 17
  Driver Configuration · 11
  LabVIEW Configuration · 15
  Required Steps · 10
I-Variable
  Numeric types · 58
  Organization · 57
I-Variables · 153
  Communication configuration · 45
  Required Communication Configuration · 19
  Specific
    Ix03 · 152
    Ix05 · 68, 81

Ix07 · 163
Ix08 · 133, 152, 163
Ix09 · 69, 82, 134, 135, 220
Ix10 · 154
Ix13 · 154
Ix14 · 154
Ix19 · 153
Ix20 · 153
Ix21 · 153
Ix22 · 193
Ix23 · 153
Ix25 · 59, 79, 153, 155, 157, 159, 162
Ix26 · 153, 155, 164, 167
Ix30 · 163
Ix60 · 134, 135
VIs to access · 29

J

Jogging and Homing Acceleration Time for Motor x (Ix20) · 153
Jogging and Homing S-Curve Time for Motor x (Ix21) · 153
Jogging Moves · 153
JOPTO Port · 184

L

LabVIEW
Clusters · ***37***
Dataflow · ***34***
General Techniques · 34
Installing PMACPanel View · 15
Persistent VI State · ***36***
Reentrancy · ***35***
Sequences · ***34***, ***35***
Supported versions · 24
Switch action · ***36***

M

Manual Layout · 1
Maximum Permitted Motor Jog Acceleration for Motor x (Ix19) · 153
Memory
VIs to access · 29
Microsoft
Visual C++ 5.0 · 9
Multi-threading · 43
M-Variable Definitions · 185
M-Variables · 173, 185

N

Naming Conventions · 39
NI-DAQ · 32
Numeric conversions
VIs to support · 52
Numeric Data Types · 27, 29

P

P (Report Motor Position) · 14
PComm32 · 10, 32
PMACPanel Interfaces · *See* PMACPanel Organization
Pewin32 · 23
PLC
VIs to access · 31
Plotting
VIs to access · 32
PmacAcc · 30
PmacAccMachineInput8 · 98, 99
PmacAccMachineOutput8 · 98, 99, 146
PmacAccMachineInput8 · 98
PmacAddress · 32, 127, 131, 236
PmacAddressAdd · 131
PmacAddressDelete · 132
PmacAddressMotors · 131, 132, 133, 135
PmacButt
PmacButtGetBool · 53
PmacButtGetDbl · 53
PmacButtGetLong · 53
PmacButtGetShort · 53
PmacButtGetStr · 50, 51, 53
PmacButtGetULong · 53
PmacButtGetUShort · 53
PmacButtSendStr · 50, 51, 53, 72
PmacButton · 50, 53
PmacButtons · 29
PmacCIN · 32
PmacCINBase · 197
PmacComm · 29, 44
PmacCommAppend · 48
PmacCommBuffer · 48
PmacCommGetBuffer · 20, 47
PmacCommGetStr · 19, 20, 44, 47
PmacCommGlobal · 48
PmacCommRespStr · 19, 20, 44, 46, 47, 51, 75, 108
PmacCommSendStr · 19, 44, 75
PmacCommGetStr · 46
PmacCoord · 30
PmacCoordColor · 70, 94
PmacCoordCurrent · 112
PmacCoordDef · 84, 91
PmacCoordIVar · 94
PmacCoordMotor2Coord · 69
PmacCoordMotorDef · 91
PmacCoordMotorsToCoord · 83, 92
PmacCoordMotorToCoord · 83, 92, 94, 168
PmacCoordMotorToEncoder · 92
PmacCoordScale · 91
PmacCoordSpecify · 93
PmacCoordStat · 55, 95
PmacCoordStatProg · 95
PmacDAQ · 32
PmacDAQMove · 188, 189, 192, 193
PmacDAQSync · 191, 192
PmacDAQSyncServo · 191, 193

PmacDAQTrigger · 191, 192
PmacPosCompGen · 190
PmacDevice · 29
PmacDevClose · 42
PmacDevOpen · 17, 18, 20, 21, 26, 42, 43, 112, 201
PmacDocument · 33
PmacDPR
PmacDPR · 32
PmacDPRVarBack · 198
PmacDPRFixedBack · 201, 218, 219, 220, 235
PmacDPRFixedBackConfig · 220
PmacDPRFixedBackVectors · 198, 218, 221
PmacDPRMotorVecToCoord · 213
PmacDPRNumericCINCluster · 231
PmacDPRNumericDbl · 226
PmacDPRNumericDWord · 225
PmacDPRNumericDWordBit · 227
PmacDPRNumericDWordBitTest · 227
PmacDPRNumericDWordSetMask · 228
PmacDPRNumericSlaveCluster · 235
PmacDPRNumericSpec · 227
PmacDPRNumericWord · 226
PmacDPRRealTimeConfig · 205, 206
PmacDPRRealTimeExamp · 211, 213
PmacDPRRealTimeMotor · 202, 205, 208, 211, 216
PmacDPRRealTimeMotors · 202, 205, 206, 213, 234, 235
PmacDPRRealTimePVE · 205, 206
PmacDPRRealTimeServo · 211
PmacDPRRealTimeVectors · 211, 212, 213, 214, 217, 218
PmacDPRVarBack · 201, 235, 236, 238, 239, 240
PmacDPRVarBackConfig · 239
PmacDPRVarBackVectors · 240
PmacEncode
PmacPosCompSetup · 175
PmacEncoder · 30
PmacEncoderCaptureExamp · 170
PmacEncoderCompare · 179
PmacEncoderCompareConfig · 177
PmacEncoderCompareExamp · 176
PmacEncoderIVarCapture · 157, 158, 159, 171
PmacEncoderOffset · 155, 167, 168
PmacEncoderPositionExamp · 166
PmacEncoderRegADC · 181
PmacEncoderRegCapture · 181
PmacEncoderRegDAC · 181
PmacEncoderRegisters · 181
PmacEncoderRegServo · 169, 181
PmacEncoderRegStat · 181
PmacEncoderRegTime · 181
PmacEncoderStatFlags · 160
PmacEncoderToCoord · 168, 169
PmacEncoderToEncoder · 168
PmacEncoderTrigger · 171, 178, 179, 180
PmacPosCompGen · 175, 176, 178, 179, 180
PmacPosCompSetup · 175, 176, 178, 179
PmacFile · 31
PmacFileDatalog · 144, 146
PmacFileDatalogAppend · 145
PmacFileDatalogCreate · 145
PmacFileDatalogRead · 145
PmacGather · 32
PmacGatherCollect · 130
PmacGatherSelect · 127, 129, 132, 237
PmacGatherSetup · 129
PmacGatherSpec · 236, 240
PmacGatherSpreadsheet · 130
PmacGatherStart · 130
PmacGatherStep · 130
PmacGatherStop · 130
PmacGlobal · 30
PmacGlobalBufferSize · 86, 89, 110
PmacGlobalControl · 86, 87, 88
PmacGlobalIVarComm · 86
PmacGlobalIVarMove · 87
PmacGlobalStat · 55
PmacGlobalStatBuffer · 87
PmacGlobalStatGather · 87
PmacGlobalStatWord1 · 89
PmacGlobalStatWord2 · 90
PmacHome · 30
PmacHomeComplete · 156, 162
PmacHomeExamp · 155
PmacHomeIVar · 155, 157, 171
PmacHomePLC1 · 164, 165
PmacHomeState · 162
PmacInc · 32
PmacInterrupt
PmacInterruptConfig · 241
PmacInterruptExamp · 241
PmacIVar · 29
PmacIVar
PmacIVarGetLong · 59
PmacIVarBool · 59
PmacIVarDbl · 59
PmacIVarGetBool · 59
PmacIVarGetDbl · 59
PmacIVarGetLong · 58
PmacIVarGetShort · 59
PmacIVarLong · 58
PmacIVarSetBool · 59
PmacIVarSetDbl · 59
PmacIVarSetLong · 58
PmacIVarSetShort · 59
PmacIVarShort · 59
PmacMemory · 29
PmacMemoryGet · 62, 98
PmacMemoryGetBit · 62
PmacMemoryGetBits · 62
PmacMemoryRead · 62
PmacMemoryReadDbl · 63
PmacMemorySet · 63, 98, 99
PmacMemorySetBit · 63
PmacMemorySetBits · 63
PmacMemoryWrite · 62
PmacMemoryWriteDbl · 63, 64
PmacMotor · 30, 68, 73, 81, 89, 151
PmacMotorCurrent · 112
PmacMotorError · 69
PmacMotorIVarFlag · 79, 80, 157, 159
PmacMotorIVarMove · 78, 155
PmacMotorIVarPID · 78, 80
PmacMotorIVarSafety · 78, 79, 80, 155
PmacMotorJog · 74
PmacMotorJogControl · 38, 39, 71, 72, 73, 84, 114

PmacMotorLimitControl · 73, 160
PmacMotorPosition · 68
PmacMotorPVE · 70, 72, 73, 74, 82, 203
PmacMotorStat · 55, 76
PmacMotorStatJog · 37, 73, 76
PmacMotorStatLimit · 73, 160
PmacMotorVelocity · 69
PmacMotors · 30
PmacMotorPosition · 69
PmacMotorsCloseLoop · 128
PmacMotorsErrors · 82
PmacMotorsPlotSelect · 84, 85
PmacMotorsPositions · 81, 82, 84
PmacMotorsPVE · 82, 83, 84, 93
PmacMotorsVelocities · 82
PmacMotorsPlotSelect · 126
PMACPanel
Documentation · 33
Panel and VI pairs · ***38***
PMACPanel Organization · 28
Communication · 29
Device Management · 26, 29
Indicators, Controls, and VIs (ICVs) · 29
Program Compilation and Download · 27
Query/Response · 26, 29
PmacPLC · 31
PmacPLCExec · 113
PmacPLCSelect · 110, 112
PmacPlot · 32
PmacPlotXYChartBuffer · 126
PmacPQM · 31
PmacPQMArray · 139, 144
PmacPQMBool · 143
PmacPQMDbl · 143
PmacPQMExamp · 139, 146, 147, 148
PmacPQMLong · 142, 143
PmacPQMLong2Var · 144
PmacPQMShort · 143
PmacPQMVar2Long · 144
PmacPQMVariant · 143, 144, 146, 148
PmacProg
PmacProgDebug · 120
PmacProgEdit · 109, 112, 113
PmacProgExec · 120
PmacProgParse · 116
PmacProgRun · 139
PmacProgSelect · 110, 112
PmacProgram · 31
PmacProgSubVI · 136
PmacResp · 52
PmacRespGetBool · 53
PmacRespGetDbl · 52
PmacRespGetLong · 53, 59
PmacRespGetShort · 53
PmacRespGetULong · 53
PmacRespGetUShort · 53
PmacResponse · 29, 53, 59
PmacSetup · 33
PmacSubVI · 31
PmacPLCSubVI · 165
PmacProgSubVI · 118, 126, 136, 137, 144, 146, 148, 149
PmacProgSubVICreate · 117
PmacTerminal
PmacTerminal · 26, 31, 44, 102, 107, 109, 110, 114, 115, 167, 170, 176, 177
PmacTerminalCoordIVars · 102
PmacTerminalEdit · 102, 115, 118, 126, 136, 137, 139, 144, 146, 147, 165, 176
PmacTerminalExecute · 102, 118, 137
PmacTerminalGather · 102, 122, 126, 237, 238
PmacTerminalGlobal · 103
PmacTerminalJog · 102, 103, 105, 114
PmacTerminalMenu · 104, 105, 106, 112, 122
PmacTerminalMotorIVars · 102
PmacTerminalMotors · 102, 122, 124, 125, 126
PmacTerminalMotorsX-Y · 102
PmacTest · 32
PmacTestCircle · 147
PmacTestCircles · 147
PmacTestExamp · 147, 149
PmacTestPQM1 · 147
PmacTestPQM1Panel · 148, 150
PmacTestPQM2 · 147
PmacTutor
PmacTutor
10 · 125
6 · 78
7 · 83, 114
PmacTutor1 · 42
PmacTutor10 · 81, 122
PmacTutor11 · 86
PmacTutor12 · 91
PmacTutor13 · 94
PmacTutor14 · 97
PmacTutor15 · 99
PmacTutor2 · 44
PmacTutor2a · 47
PmacTutor3 · 47, 50, 52, 53
PmacTutor4 · 52, 55, 59
PmacTutor5 · 55, 76
PmacTutor6 · 57
PmacTutor6b · 61
PmacTutor7 · 68
PmacTutor8 · 73
PmacTutor9 · 78
PmacTutor1 · 41
PmacTutor2 · 41
PmacTutorApp · 41
PmacTutorial · 32
PmacTutorSub · 41
PmacUtility · 33
Position
Querying · 52
Position Capture · 30
Position Extension in Software · 151
Position Loop Feedback Address for Motor x (Ix03) · 152
Position Processing
Software Position Extension · 151
Position Scale Factor for Motor x (Ix08) · 152
Position-Capture Function · 153, 155, 164, 169
Setting the Trigger Condition · 153
Using for Homing · 153
Using in User Program · 169
Position-Compare Function

Compare Control Bits · 173
Directly Triggering External Action · 175
Preloading the Compare Position · 173
Required M-Variables · 173
Position-Compare Outputs
PMAC-Lite · 183
PMAC-PC · 183
PMAC-STD · 184
PMAC-VME · 183
PQM Variables
VIs to access · 31
Preloading the Compare Position · 173
Programs
VIs to access · 31
PTalk · 23
PTalk Active X · 24

S

Safety
Electrical · 4
Motor Movement · 4
Program download · 3
Serial communication · 43
Servo Interrupt Time Variable (I10) · 14
Sinking Inputs · 185
Sinking Outputs · 185
Sourcing Inputs · 185
Sourcing Outputs · 185
Status
Querying · 55
Storing the Home Position · 154
Supported PMAC Models · 7
Synchronizing To External Events
Position-Capture · 153
Position-Compare · 172
Synchronous M-Variable Assignment · 185

T

Technical Support · 5
Terminal Conventions · 40
Triggering · 30
Triggering External Action · 175
Trouble Shooting
Driver Communication · 20
Tutorials · 32, 41
Accessing PMAC · 42
Communicaiton logging · 47
I-Variable access · 57
Numeric responses · 52
Query/Response · 44
Sending Commands · 50, 52
Status · 55

V

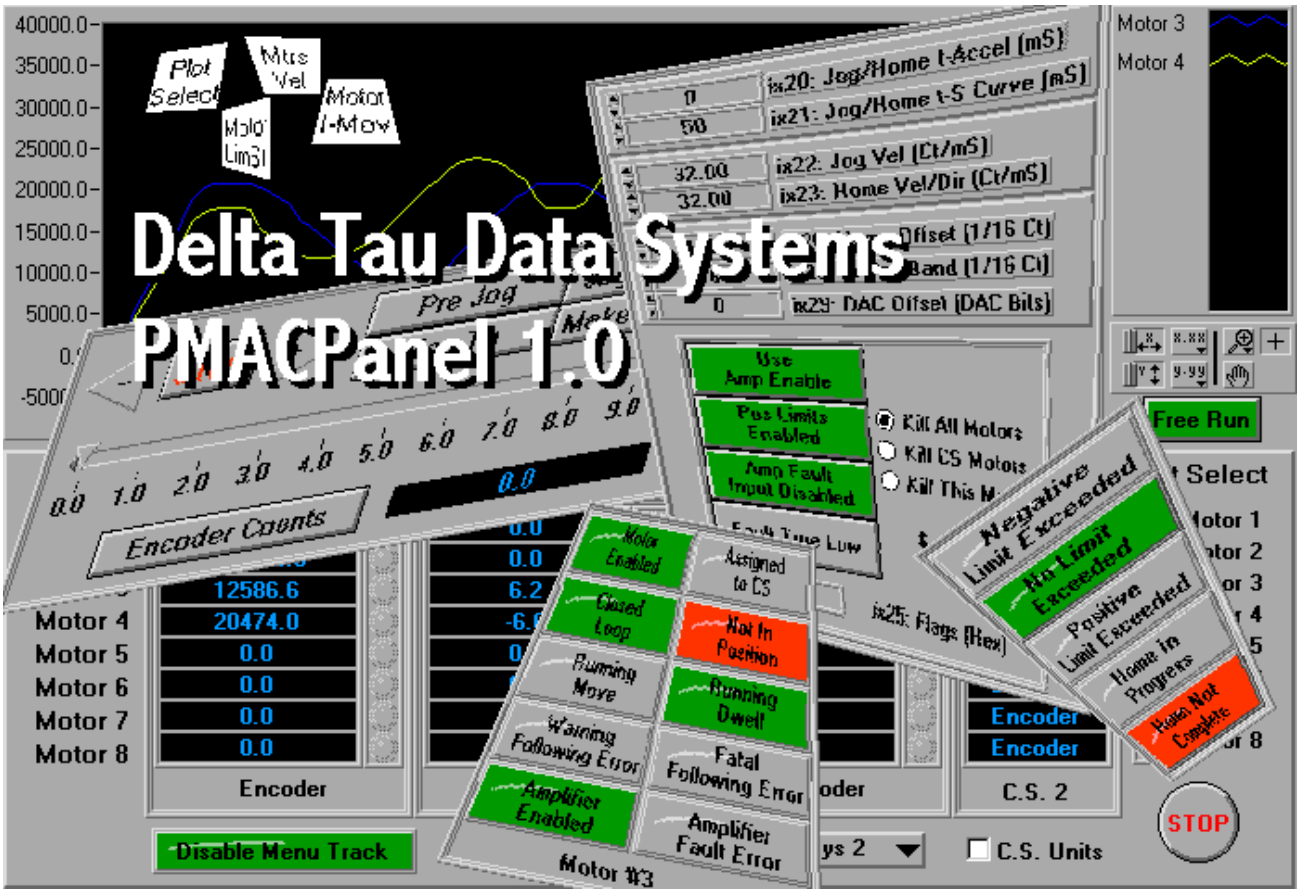
Velocity
Querying · 53
VI Compilation · 16

Z

Homing · 155
Zero-Move Homing · 155

PMAC Motion Control for LabVIEW 5.0

PMACPanel VI Reference



By Delta Tau Data Systems, Inc.

Contents

Chapter 13 - VI Reference 3

Basics 3
PmacAcc 4
VIs 5
PmacAddress 6
VIs 6
Indicator and Control Clusters 8
PmacButton 8
VIs 8
PmacCIN 12
PmacComm 12
VIs 12
Indicator and Control Clusters 15
PmacCoord 16
VIs 16
Indicator and Control Clusters 22
PmacDAQ 36
PmacDevice 36
VIs 36
PmacDPR 37
VIs 37
Indicator and Control Clusters 54
PmacEncoder 67
VIs 67
Indicator and Control Clusters 77
PmacFile 86
VIs 87
Indicator and Control Clusters 92
PmacGather 93
VIs 93
Indicator and Control Clusters 97
PmacGlobal 99
VIs 99
Indicator and Control Clusters 102
PmacHome 117
VIs 117
Indicator and Control Clusters 119
PmacIVar 133
VIs 133
PmacMemory 139
VIs 139
PmacMotor 143
VIs 143
Indicator and Control Clusters 151
PmacMotors 178

VIs 178
Indicator and Control Clusters....181
PmacPLC....182
VIs 182
Indicator and Control Clusters....187
PmacPlot....188
VIs 188
PmacPQM 189
VIs 189
Indicator and Control Clusters....195
PmacProgram 197
VIs 198
Indicator and Control Clusters....204
PmacResponse....210
VIs 210
PmacSubVI 212
VIs 212

Appendix A - Compiling CINs Using Microsoft Visual C++ 215

Introduction....215
How do I Compile a Code Interface Node (CIN) using Visual C++ 5.0? 215

Glossary of Terms 219

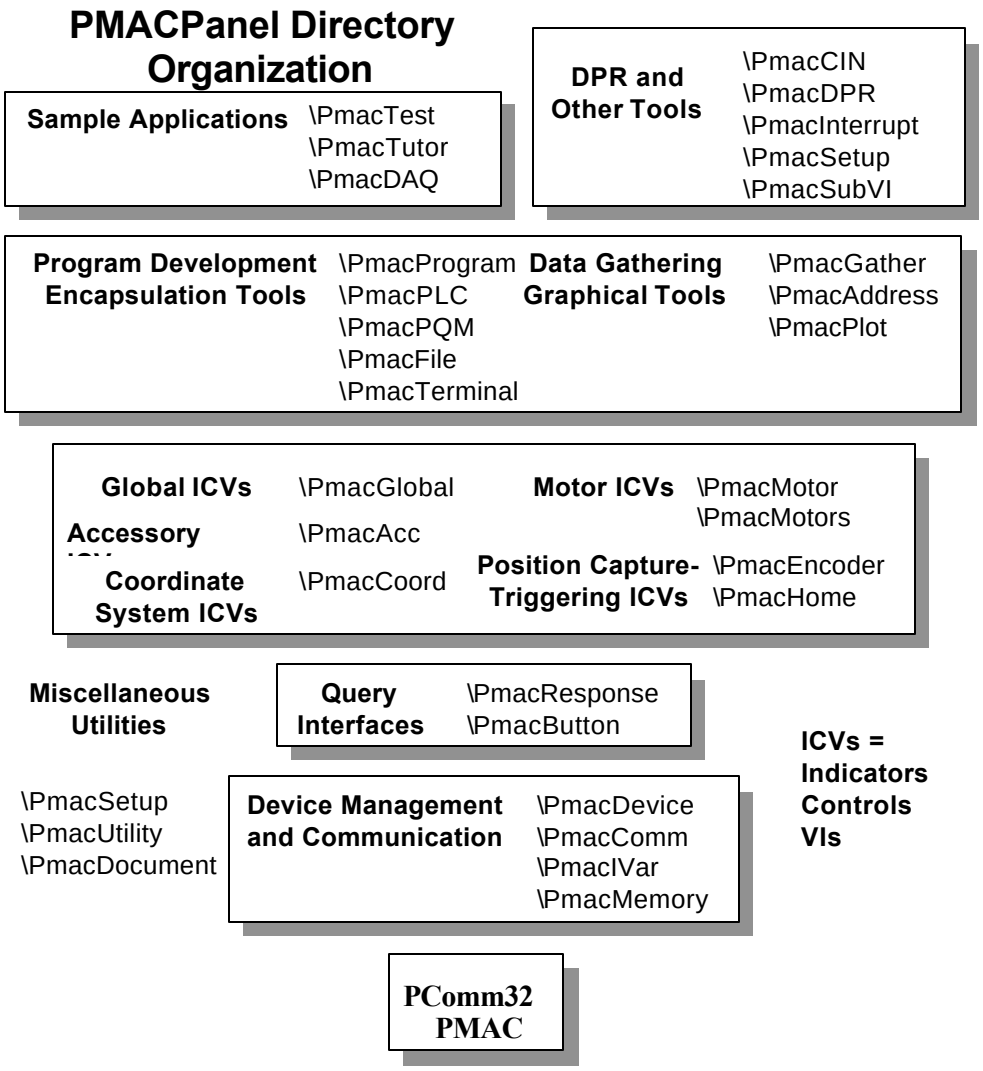
Index 220

Chapter 13 - VI Reference

Basics

This brief explanation of PMACPanel's organization will help you to get the information you need quickly and painlessly as well as help you plan your application's architecture.

The PMACPanel library, contained in the directory PMACPanel.lib, is divided into five basic categories as illustrated in the following figure. These categories provide an increasing level of capability as you progress from the lower levels to the higher levels.



Within each category are several sub-directories containing collections of indicators, controls, and VIs (ICVs) that provide capabilities to make your application development task easier. The VIs in each sub-directory follow a naming convention that makes them easier to locate. For example, in the directory **\PmacCoord** all VIs are named **PmacCoord ….** The names of VIs in a given sub-directory may have shortened names to make them a little easier to fit on screens and such. For example, VIs in **\PmacResponse** are shortened to **PmacResp….** The collections of VIs in each category implement most commonly used capabilities and generally have examples to demonstrate their use. The examples will have the word **Examp** or **Example** at the end of their name. The remaining chapters cover numerous tutorial exercises and examples to demonstrate their use.

PmacAcc

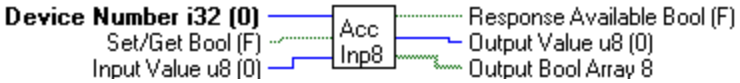
\PmacAcc - PMAC has numerous accessories that are used to provide digital I/O, analog I/O, and control capabilities. This includes the ACC16D front panel and the

alphanumeric display. This collection provides basic ICVs that access the memory mapped data used by these accessories.

VIs

PmacAccMachineInput8

If Set\Get is FALSE or not wired get the Machine Input Port contents. Response Available will be TRUE to indicate the Outputs contain the value. If Set\Get is TRUE set the Machine Input Port using Input Value. Response Available will be FALSE and the outputs default to Input Value.



Device Number i32 (0) Provided by PmacDevOpen

Input Value u8 (0) Input Value for Set operation

Set/Get Bool (F) Set Machine Input Port when TRUE. Get Machine Input Port when FALSE or unwired.

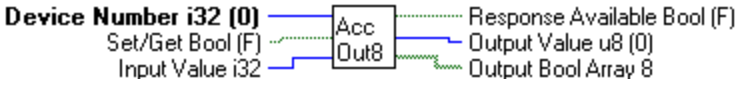
Output Value u8 (0) Value of Machine Input Port during Get or Set operation.

Output Bool Array 8 Boolean Array representation of Machine Input Port during Get or Set operation.

Response Available Bool (F) TRUE when Get operation produces valid Response

PmacAccMachineOutput8

If Set\Get is FALSE or not wired get the Machine Output Port contents. Response Available will be TRUE to indicate the Outputs contain the value. If Set\Get is TRUE set the Machine Output Port using Input Value. Response Available will be FALSE and the outputs default to Input Value.



Device Number i32 (0) Provided by PmacDevOpen

Input Value i32 Input Value for Set operation

Set/Get Bool (F) Set Machine Output Port when TRUE. Get Machine Output Port when FALSE or unwired.

Output Bool Array 8 Boolean Array representation of

Machine Output Port during Get or Set operation.

Output Value u8 (0) Value of Machine Output Port during Get or Set operation.

Response Available Bool (F) TRUE when Get operation produces valid Response

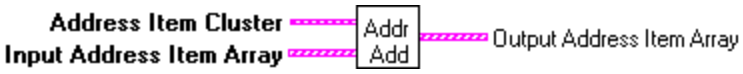
PmacAddress

\PmacAddress - A collection of tools for specifying addresses, scale factors, and descriptions for gathering. Variables not already accessible from PMACPanel can easily be added to the tables.

VIs

PmacAddressAdd

Check to see if the item specified by Address Item Cluster already exists in Input Address Item Array. If it already exists do not add it. If it does not exist add the cluster to Output Address Item Array.



Input Address Item Array Array of Address Item clusters to add Address Item to.

Address Item Cluster Specify a description, address, and scale factor for a Address Item

Address Item Cluster Specify a description, address, and scale factor for a Address Item

Address Item Description Text description of Address Item

Address Item Address Address of Address Item

Address Item Scale Scale factor for Address Item

Output Address Item Array Array of Address Item clusters Address Item is added to if it doesn't already exist.

Address Item Cluster Specify a description, address, and scale factor for a Address Item

PmacAddressDelete

Locate and remove the Address Item Cluster specified by Selection Index to Delete from the Input Address Item Array.

Selection Index to Delete i32 Ring selection index for to delete from table

Input Address Item Array Array of Address Item clusters to delete Address Item indexed by Selection Index to Delete from.

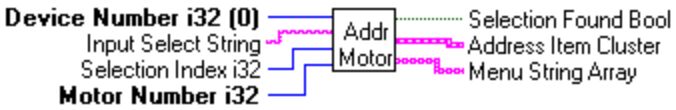
Address Item Cluster Specify a description, address, and scale factor for a Address Item

Output Address Item Array Array of Address Item clusters after indexed Address Item is deleted.

Address Item Cluster Specify a description, address, and scale factor for a Address Item

PmacAddressMotors

This VI maintains a table defining 29 of the most common Address Items. If Input Select String is the empty string the VI produces Menu String Array describing the defined Address Items. This should be used to set the items in a Menu or Text ring control. Selection Index and Motor Number are provided by rings and define the desired item and the motor number used to compute an address for the specified item. The computed item is contained in Address Item Cluster. For a description of this computation see the reference section and the memory map contained in the PMAC Software Reference Manual.



Input Select String String to select entry in Translation Table

Selection Index i32 Ring selection index for Translation Table

Motor Number i32 Used to address a specific motor for translating address

Device Number i32 (0) Provided by PmacDevOpen

Menu String Array String array for setting menu or text rings with Translation Table items

Selection Found Bool TRUE when a selection is found and Gather Address Cluster can be used

Address Item Cluster Specify a description, address, and scale factor for the selected Address Item

Address Item Description Text description of **Address Item**

Address Item Address Address of Address Item

Address Item Scale Scale factor for Address Item

Indicator and Control Clusters

PmacAddressItem

This cluster is used to define a Address Item's text description, address, and scale factor.

Address Item Cluster Specify a description, address, and scale factor for a Address Item

Address Item Description Text description of Address Item

Address Item Address Address of Address Item

Address Item Scale Scale factor for Address Item

Address Item Type Enumerated type defining type of raw data

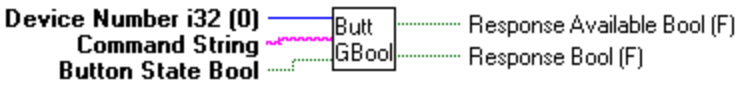
PmacButton

\PmacButton - In most user interfaces queries are sent to PMAC as the result of Boolean user actions and conditions. This group of VIs send command strings to request responses when an associated LabVIEW Boolean condition is TRUE.

VIs

PmacButtGetBool

Send Command String to PMAC and wait for a response when Button State is TRUE. When Response Available is TRUE Response Bool contains the response. If Response Available is FALSE Response Bool defaults to FALSE.



Command String PMAC on-line command string to send to PMAC

Device Number i32 (0) Provided by PmacDevOpen

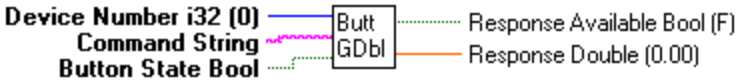
Button State Bool Command String is sent when TRUE

Response Available Bool (F) TRUE when an output response to Command String is valid

Response Bool (F) Formatted response to Command String when Button State is TRUE

PmacButtGetDbl

Send Command String to PMAC and wait for a response when Button State is TRUE. When Response Available is TRUE Response Double contains the response. If Response Available is FALSE Response Double defaults to 0.0.



Command String PMAC on-line command string to send to PMAC

Device Number i32 (0) Provided by PmacDevOpen

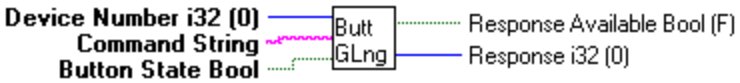
Button State Bool Command String is sent when TRUE

Response Available Bool (F) TRUE when an output response to Command String is valid

Response Double (0.00) Formatted response to Command String when Button State is TRUE

PmacButtGetLong

Send Command String to PMAC and wait for a response when Button State is TRUE. When Response Available is TRUE Response i32 contains the response. If Response Available is FALSE Response i32 defaults to 0.



Command String PMAC on-line command string to send to PMAC

Device Number i32 (0) Provided by PmacDevOpen

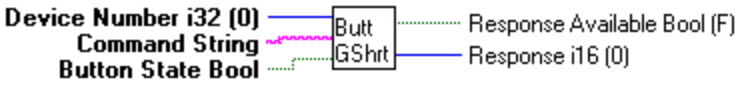
Button State Bool Command String is sent when TRUE

Response Available Bool (F) TRUE when an output response to Command String is valid

Response i32 (0) Formatted response to Command String when Button State is TRUE

PmacButtGetShort

Send Command String to PMAC and wait for a response when Button State is TRUE. When Response Available is TRUE Response i16 contains the response. If Response Available is FALSE Response i16 defaults to 0.



Command String PMAC on-line command string to send to PMAC

Device Number i32 (0) Provided by PmacDevOpen

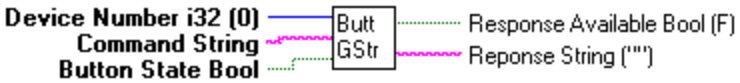
Button State Bool Command String is sent when TRUE

Response Available Bool (F) TRUE when an output response to Command String is valid

Response i16 (0) Formatted response to Command String when Button State is TRUE

PmacButtGetStr

Send Command String to PMAC and wait for a response when Button State is TRUE. When Response Available is TRUE Response String contains the response. If Response Available is FALSE Response String defaults to the empty string.



Command String PMAC on-line command string to send to PMAC

Device Number i32 (0) Provided by PmacDevOpen

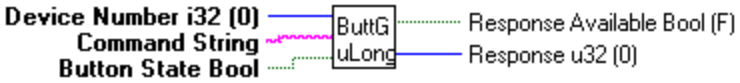
Button State Bool Command String is sent when TRUE

Response String ("") Formatted response to Command String when Button State is TRUE

Response Available Bool (F) TRUE when an output response to Command String is valid

PmacButtGetULong

Send Command String to PMAC and wait for a response when Button State is TRUE. When Response Available is TRUE Response u32 contains the response. If Response Available is FALSE Response u32 defaults to 0.



Command String PMAC on-line command string to send to PMAC

Device Number i32 (0) Provided by PmacDevOpen

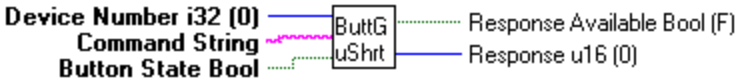
Button State Bool Command String is sent when TRUE

Response Available Bool (F) TRUE when an output response to Command String is valid

Response u32 (0) Formatted response to Command String when Button State is TRUE

PmacButtGetUShort

Send Command String to PMAC and wait for a response when Button State is TRUE. When Response Available is TRUE Response u16 contains the response. If Response Available is FALSE Response u16 defaults to 0.



Command String Command String

Device Number i32 (0) Device Number

Button State Bool Button State

Response Available Bool (F) Output Valid

Response u16 (0) Double Output Value

PmacButtSendStr

Send Command String to PMAC when Button State is TRUE. Response Available is TRUE when PMAC has processed the command.

Command String PMAC on-line command string to send to PMAC

Device Number i32 (0) Provided by PmacDevOpen

Button State Bool Command String is sent when TRUE

Response Available Bool (F) TRUE when Command String is sent to PMAC

PmacCIN

The VIs in this collection are templates for Code Interface Node development. See Chapter 10 for details.

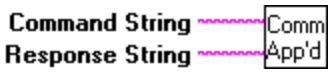
PmacComm

\PmacComm - Provide the ability to send on-line ASCII command strings and data to PMAC and receive ASCII responses from PMAC. Error handling and the ability to buffer communication for analysis and future reference are provided at this level.

VIs

PmacCommAppend

Copy Command String and Response String to the last communication items in PmacCommGlobal. If Logging Enabled is TRUE they are also appended to the Communication Log.

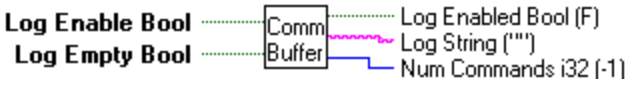


Command String Last on-line Command String sent to PMAC by PmacCommSendString or PmacCommRespStr.

Response String Last response received from PMAC in response to Command String.

PmacCommBuffer

When Log Enable is TRUE communication logging is enabled. Log Enabled Bool, Log String, and Num Commands reflect the state of the log buffer when logging is enabled. Log String is the empty string, Num Commands = -1, and Log Enabled is FALSE when logging is disabled. When Log Empty is TRUE the log is emptied.



Log Enable Bool Enable logging.

Log Empty Bool Empty the log buffer

Log String ("") Contents of the Log Buffer when Log

Enabled is TRUE. Empty string otherwise.

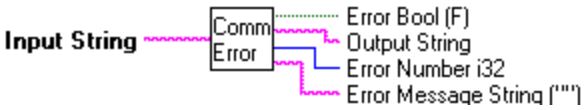
Num Commands i32 (-1) Number of commands in the Log buffer when Log Enabled is TRUE. -1 otherwise.

Log Enabled Bool (F) TRUE when Logging is enabled

PmacCommError

Parse Input String for PMAC ERR codes. This requires proper communication I-Variable configuration - see the PMACPanel manual. If an ERR code is found it is decoded and a modal dialog is displayed telling you the command that caused the error, a description of the error, and possible remedies. The user can decide to Abort execution of the application VI or ignore the error and Continue. With or without errors Output String is a copy of Input String after parsing.

If an error is found Error Bool is TRUE. Error Number and Error Message contain the error code and text description of the error. If no error is found Error Bool is FALSE. Error Number = -1 and Error Message is "No Error".



Input String PMAC response strings to parse for ERR codes

Output String Input String after parsing for ERR codes

Error Message String ("") ERR description for found errors

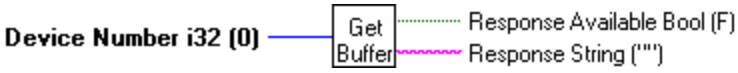
Error Number i32 PMAC ERR code if error is found. -1 if no error.

Error Bool (F) TRUE when an ERR code is found in Input String

PmacCommGetBuffer

Check if PMAC has data available. When Response Available is TRUE Response String contains all available data. When Response Available is FALSE Response String is the empty string. Responses are not parsed for PMAC ERR codes.

This VI is used by PmacCommGetResponse and PmacCommGetStr to retrieve very large response buffers. These two VIs are limited by PComm32 to responses of 256 characters whereas this VI is not. You will not generally use this VI.



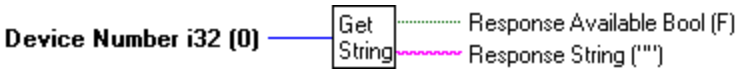
Device Number i32 (0) Provided by PmacDevOpen

Response String ("") Extended response buffer retrieved from PMAC

Response Available Bool (F) TRUE when to Response String is valid

PmacCommGetStr

Check if PMAC has data available. When Response Available is TRUE Response String contains all available data. When Response Available is FALSE Response String is the empty string. Responses are parsed for PMAC ERR codes and flagged with a modal dialog.



Device Number i32 (0) Provided by PmacDevOpen

Response String ("") Unsolicited PMAC data

Response Available Bool (F) TRUE when to Response String is valid

PmacCommRespStr

Send Command String to PMAC and wait for a response. If Command String is the empty string nothing is sent. Response Available is TRUE when Response String contains response data. When Response Available is FALSE Response String is the empty string. Responses are parsed for PMAC ERR codes and flagged with a modal dialog.

Device Number i32 (0) Provided by PmacDevOpen

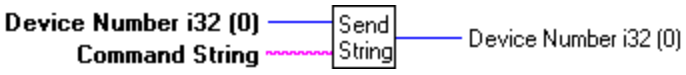
Command String PMAC on-line command string to send to PMAC

Response String ("") PMAC response to Command String

Response Available Bool (F) TRUE when to Response String is valid

PmacCommSendStr

Send Command String to PMAC. If Command String is the empty string nothing is sent. The output Device Number is a copy of input Device Number to allow sequencing of commands to PMAC.



Command String Command String

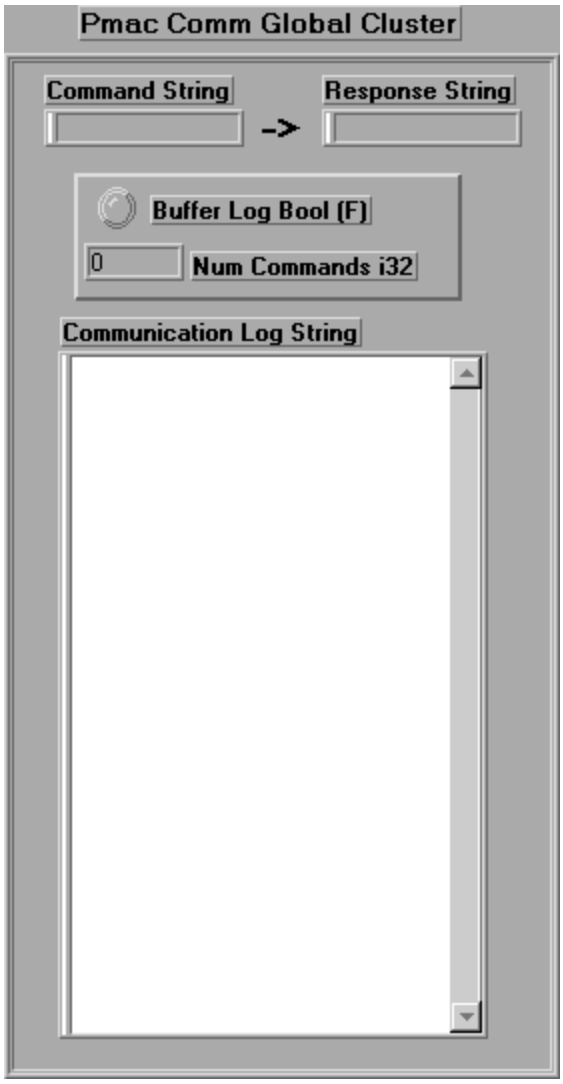
Device Number i32 (0) Provided by PmacDevOpen

Device Number i32 (0) Passed from input Device Number after Command String is processed

Indicator and Control Clusters

PmacCommGlobal

This cluster maintains the last communication with PMAC and logging capability



Pmac Communication Cluster This cluster maintains a log of communication between PMACPanel and PMAC.

Command String Last on-line command sent to PMAC

Response String Last response received from PMAC

Communication Log String A multi-line buffer of commands sent to PMAC and received from PMAC.

Num Commands i32 The number of commands sent to PMAC and logged in the Communication Buffer.

Buffer Log Bool (F) When TRUE all communication is appended and logged to the Communication Log.

PmacCommLast

This cluster maintains a copy of the last communication with PMAC. It is used to aid in error handling.

Last Pmac Communication Cluster This cluster maintains a copy of the last on-line command sent to PMAC and response received from PMAC.

Command String Last on-line command sent to PMAC

Response String Last response received from PMAC

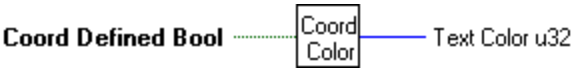
PmacCoord

\PmacCoord - Monitor the execution of programs and definition of motor coordinate systems. This information is required for the development of user interfaces in which information is entered and displayed in coordinate system units rather than encoder units.

VIs

PmacCoordColor

Output a color constant for coloring Numeric or Text controls in PMACPanel indicator clusters for position, velocity, and following errors. The color is Orange when Coordinate Define is TRUE and Blue when FALSE. It should be tied to an attribute node for the desired control.

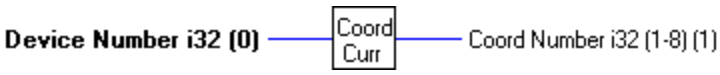


Coord Defined Bool The desired CS is defined.

Text Color u32 Color for setting attributes on controls and indicators.

PmacCoordCurrent

Query PMAC for the currently addressed coordinate system. It is most generally used in interactive development environments rather than a custom VI where you want to address a specific CS.



Device Number i32 (0) Provided by PmacDevOpen

Coord Number i32 (1-8) (1) Currently addressed coordinate system in PMAC.

PmacCoordDef

Fetch the motor scaling definitions for the specified coordinate system and provide a cluster for the PmacCoordDef indicator.

Device Number i32 (0) Provided by PmacDevOpen

Coord Number i32 (1-8) (1) CS Number to fetch the definition for.

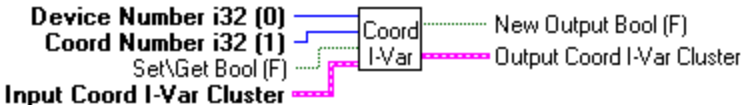
Coordinate System Definition Cluster

Definition Array Array of strings for Coordinate System definitions

C.S. Defined Indicator specifying which CS is defined

PmacCoordIVar

Follow PMACPanel's standard I-Variable VI architecture. When Set/Get is TRUE the I-Variables for the specified Coord Number are set. Otherwise they are fetched from PMAC and provided by Output Coord I-Var Cluster with New Output TRUE. Set/Get is not required and defaults to a Get operation.



Coord Number i32 (1) Coordinate system number whose I-Variables are addressed

Device Number i32 (0) Provided by PmacDevOpen

Input Coord I-Var Cluster Input PmacCoordIVar cluster for most common CS I-Variables

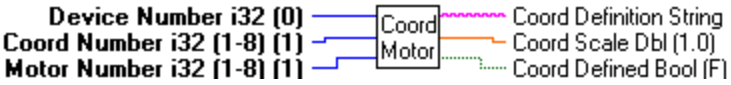
Set\Get Bool (F) When TRUE PMAC is set using Input Coord I-Var Cluster. When FALSE Output Coord I-Var Cluster contains the current PMAC definitions.

New Output Bool (F) TRUE when Output Coord I-Var Cluster contains new data.

Output Coord I-Var Cluster Output PmacCoordIVar cluster for most common CS I-Variables

PmacCoordMotorDef

Query PMAC for the definition of Motor Number in Coord Number. If the motor is not defined in the specified CS Coord Definition is "Encoder", Coord Scale = 1.0, and Coord Defined is FALSE. If the motor is defined in the specified CS Coord Defined is TRUE, Coord Scale is the encoder to CS unit scale factor, and Coord Definition is the definition (e.g. "#1->16000X").



Device Number i32 (0) Provided by PmacDevOpen

Coord Number i32 (1-8) (1) Coordinate system number we expect the Motor Number to be assigned to.

Motor Number i32 (1-8) (1) Motor number whose coordinate system definition we are trying to determine.

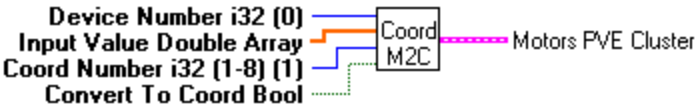
Coord Defined Bool (F) TRUE if the specified motor is defined in the specified CS.

Coord Definition String String defining the motor's definition in the CS or "Encoder" if not defined.

Coord Scale Dbl (1.0) Encoder to CS unit scale factor for motor. 1.0 if motor is not defined in CS.

PmacCoordMotorsToCoord

Generate an indicator cluster for PmacMotorsPVE.ctl. Input Value Double is an array of positions, velocities, or following errors from VIs in the PmacMotors collection. If Convert To Coord is TRUE fetch the CS definitions for the motors specified in Coord Number and scale them to CS units. Motors not defined in Coord Number are not scaled.



Input Value Double Array Array of motor positions, velocities, or following errors given in encoder counts by VIs in PmacMotors to possibly convert to CS units.

Convert To Coord Bool Convert states for motors defined in the specified CS to CS units.

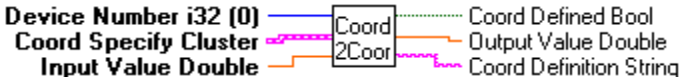
Coord Number i32 (1-8) (1) Coordinate number to use when converting the array of motor states to CS units.

Device Number i32 (0) Provided by PmacDevOpen

Motors PVE Cluster The indicator cluster displays an array of values for all PMAC motors. The array may be positions, velocities, or following errors. The array of Boolean indicators indicate which values are in CS units. The caption specifies the displayed values as being in encoder counts or a specific CS.

PmacCoordMotorToCoord

Coord Specify Cluster specifies a motor within a CS and an attempt to convert Input Value from encoder counts to CS units. If the motor is not defined in the CS no conversion is applied. If the motor is defined and Convert is TRUE Coord Defined is TRUE and Output Value is scaled from encoder counts to CS units. Coord Definition is a string specifying Output Value units as "Encoder" or the CS definition of the motor.



Input Value Double Motor position, velocity, of following error from PmacMotor VIs to possibly convert to CS units

Device Number i32 (0) Provided by PmacDevOpen

Coord Specify Cluster Cluster defining the motor, CS, and conversion state to be applied

Output Value Double Input value in encoder units converted to CS units if the motor is defined in the CS. Otherwise equal to Input Value.

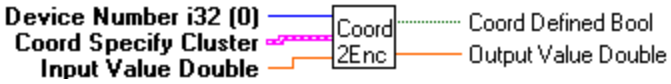
Coord Defined Bool If Motor Number is defined in Coord Number and Convert Bool is TRUE this is TRUE.

Coord Definition String String defining the motor within the CS. "Encoder" if not defined.

PmacCoordMotorToEncoder

Coord Specify Cluster specifies a motor within a CS and an attempt to convert Input Value from CS units to encoder counts. If the motor is not defined in the CS no

conversions is applied. If the motor is defined and Convert is TRUE Coord Defined is TRUE and Output Value is scaled from CS units to encoder counts.



Input Value Double Motor position, velocity, or following error from PmacMotor VIs to possibly convert to encoder units

Device Number i32 (0) Provided by PmacDevOpen

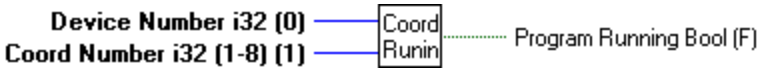
Coord Specify Cluster Cluster defining the motor, CS, and conversion state to be applied

Output Value Double Input value in CS units converted to encoder counts if the motor is defined in the CS. Otherwise equal to Input Value.

Coord Defined Bool If Motor Number is defined in Coord Number and Convert Bool is TRUE this is TRUE.

PmacCoordRunning

Determine whether there is a program running in Coord Number.



Device Number i32 (0) Provided by PmacDevOpen

Coord Number i32 (1-8) (1) Coordinate system to check for a running program

Program Running Bool (F) When TRUE there is a program running in Coord Number.

PmacCoordScale

Query PMAC for the motors defined in Coord Number. The Coordinate System Scale Cluster (PmacCoordScale.ctl) contains three arrays with the motor definition, scale factor, and whether or not the motor is defined in Coord Number. The actual query is only placed if Coord Number changes from a previous call.

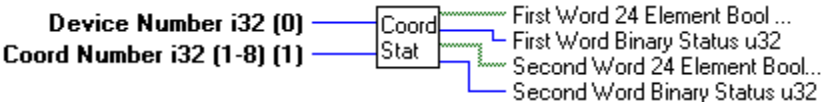
Device Number i32 (0) Provided by PmacDevOpen

Coord Number i32 (1-8) (1) CS to fetch the definitions for

Coordinate System Scale Cluster

PmacCoordStat

Query PMAC for the status of the CS specified by Coord Number. Report the two status words as arrays of Booleans and unsigned 32 bit integers.


Device Number i32 (0)
Coord Number i32 (1-8) (1)
Coord
Stat
First Word 24 Element Bool ...
First Word Binary Status u32
Second Word 24 Element Bool...
Second Word Binary Status u32

Device Number i32 (0) Provided by PmacDevOpen

Coord Number i32 (1-8) (1) CS to obtain status for

First Word 24 Element Bool Array 24 element Boolean array containing first status word

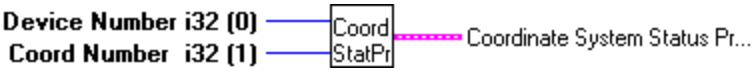
Second Word 24 Element Bool Array 24 element Boolean array containing second status word

First Word Binary Status u32 integer representation of first word status

Second Word Binary Status u32 integer representation of second word status

PmacCoordStatProg

Create a status indicator cluster for the PmacCoordStatProg indicator containing the status for Coord Number.



Device Number i32 (0) Device Number

Coord Number i32 (1) CS to obtain status for

Coordinate System Status Program Cluster Cluster for PmacCoordStatProg

PmacCoordStatWord1

Create a status indicator cluster for the PmacCoordStat1 indicator containing the status for Coord Number.

Device Number i32 (0) Provided by PmacDevOpen

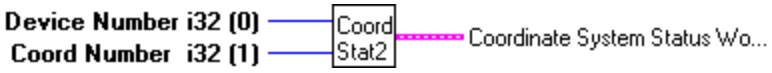
Coord Number i32 (1) CS to obtain status for

Coordinate System Status Word 1 Cluster Cluster for

PmacCoordStat1

PmacCoordStatWord2

Create a status indicator cluster for the PmacCoordStat2 indicator containing the status for Coord Number.



Device Number i32 (0) Provided by PmacDevOpen

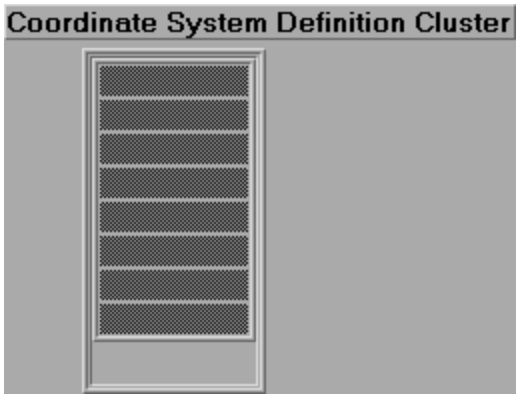
Coord Number i32 (1) CS to obtain status for

Coordinate System Status Word 2 Cluster Cluster for PmacCoordStat2

Indicator and Control Clusters

PmacCoordDef

Definitions for motor scaling in a given coordinate system are shown in this indicator. The indicator accepts the cluster provided by the PmacCoordDef VI.



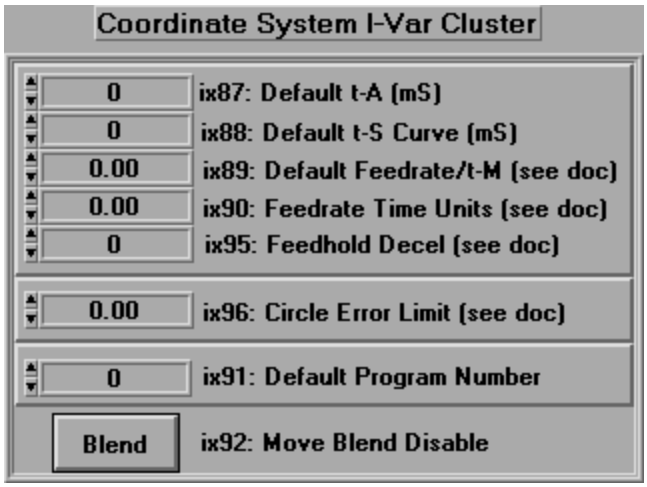
Coordinate System Definition Cluster

Definition Array Array of strings for Coordinate System definitions

C.S. Defined Indicator specifying which CS is defined

PmacCoordIVar

This cluster contains items for displaying and modifying the most common coordinate system I-Variables.



Coordinate System I-Var Cluster Cluster for most common CS I-Variables

ix87: Default t-A (mS) This parameter sets the default time for commanded acceleration for programmed blended LINEAR and CIRCLE mode moves in coordinate system x. It also provides the default segment time for SPLINE mode moves. The first use of a TA statement in a program overrides this value.

Even though this parameter makes is possible not to specify acceleration time in the motion program, you are strongly encouraged to use TA in the program and not rely on this parameter, unless you must keep to a syntax standard that does not support this (e.g. RS-274 "G-Codes"). Specifying acceleration time in the program along with speed and move modes makes it much easier for later debugging.

If the specified S-curve time (see Ix88, below) is greater than half the specified acceleration time, the time used for commanded acceleration in blended moves will be twice the specified S-curve time.

The acceleration time is also the minimum time for a blended move; if the distance on a feedrate-specified (F) move is so short that the calculated move time is less than the acceleration time, or the time of a time-specified (TM) move is less than the acceleration time, the move will be done in the acceleration time instead. This will slow down the move

The acceleration time will be extended automatically when any motor in the coordinate system is asked to exceed its maximum acceleration rate (Ix17) for a programmed LINEAR-mode move with I13=0 (no move segmentation).

Make sure that the specified acceleration time (Ix87 or 2*Ix88) is greater than zero, even if you are planning to rely on the maximum acceleration rate parameters. A specified acceleration time of zero will cause a divide-by-zero error. The minimum

specified time should be Ix87=1, Ix88-0.

ix88: Default t-S Curve (mS) This parameter set the default time in each "half" of the "S" in S-curve acceleration for programmed blended LINEAR and CIRCLE mode moves in coordinate system x. It does not affect SPLINE, PVT, or RAPID mode moves. The first use of a TS statement in a program overrides this value.
Even though this parameter makes is possible not to specify acceleration time in the motion program, you are strongly encouraged to use TS in the program and not rely on this parameter, unless you must keep to a syntax standard that does not support this (e.g. RS-274 "G-Codes"). Specifying acceleration time in the program along with speed and move modes makes it much easier for later debugging.
If Ix88 is zero, the acceleration is constant throughout the Ix87 time and the velocity profile is trapezoidal. If Ix88 is greater than zero, the acceleration will start at zero and linearly increase through Ix88 time, the stay constant (for time TC) until Ix87-Ix88 time, and linearly decrease to zero at Ix87 time (that is Ix87=2Ix88 - TC). If Ix88 is equal to Ix87/2, the entire acceleration will be spec in S-curve form (Ix88 values greater than Ix87/2 override the Ix87 value; total acceleration time will be 2*Ix88).
The acceleration time will be extended automatically when any motor in the coordinate system is asked to exceed its maximum acceleration rate (Ix17) for a programmed LINEAR-mode move with I13=0 (no move segmentation).
Make sure the specified acceleration time (TA or 2*TS) is greater than zero, even if you are planning to rely on the maximum acceleration rate parameters (Ix17). A specified acceleration time of zero will cause a divide-by-zero error. The minimum specified time should be TA1 TS0.

ix89: Default Feedrate/t-M (see doc) This parameter sets the default feedrate (commanded speed) for programmed LINEAR and CIRCLE mode moves in coordinate system x. The first use of an F or TM statement in a motion program overrides this value. The velocity units are defined by the position and time units, as defined by axis definition statements and Ix90.
You are strongly encouraged not to rely on this parameter and to declare your feedrate in the program. This will keep your move parameters with your move commands, lessening the chances of future errors, and making debugging easier.

ix90: Feedrate Time Units (see doc) This parameter defines the time units used in commanded velocities

(feedrates) in motion programs executed by coordinate system x. Velocity units are comprised of length units divided by time units. The length units are determined in the axis definition statements for the coordinate system. Ix90 sets the time units. Ix90 itself has units of milliseconds, so if Ix90 is 60,000, the time units are 60,000 milliseconds, or minutes. The default value of Ix90 is 1000 msec, specifying velocity time units of seconds.
This affects two type of motion program values: F values (feedrate) for LINEAR- and CIRCLE-mode moves; and the velocities in the actual move commands for PVT-mode moves.
Examples
If position units have been set as centimeters by the axis definition statements, and it is desired that feedrate values be specified in cm/sec, this parameter would be set to 1000.0 (time units = sec).
If position units have been set as degrees by the axis definition statements, and it is desired that feedrate values be specified in deg/min, this parameter would be set to 60,000.0 (time units = minutes).
If a spindle is rotating at 4800 rpm, with a linear axis specified in inches, and it is desired that linear speed be specified in inches per spindle revolution, Ix90 would be set to 12.5 ([1 min/4800 rev] * [60,000 msec/ min] = 12.5 msec/rev).

ix95: Feedhold Decel (see doc) This parameter controls the rate at which the axes of the coordinate system stop if a feed hold command (H) is given, and the rate at which they start up again on a succeeding run command (R or S). A feed hold command is equivalent to a %0 command except that it uses Ix95 for its slew rate instead of Ix94. Having separate slew parameters for normal time-base control and for feed hold commands allows both responsive ongoing time-base control (Ix94 relatively high) and well-controlled holds (Ix95 relatively low).
The default Ix95 value of 1644, when used on a card set up with the default servo cycle time of 442 m sec, provides a transition time between %100 and %0 (feed hold) of one second.

ix96: Circle Error Limit (see doc) In a circular arc move, a move distance that is more than twice the specified radius will cause a computation error because a proper path cannot be found. Sometimes, due to round-off errors, a distance slightly larger than twice the radius is given (for a half-circle move), and it is desired that this not create an error condition. This parameter allows the user to set an error limit on the amount the move distance is greater than twice the radius. If the move distance is greater than 2R, but by less than this limit, the move is done in a spiral fashion to the endpoint, and no error condition

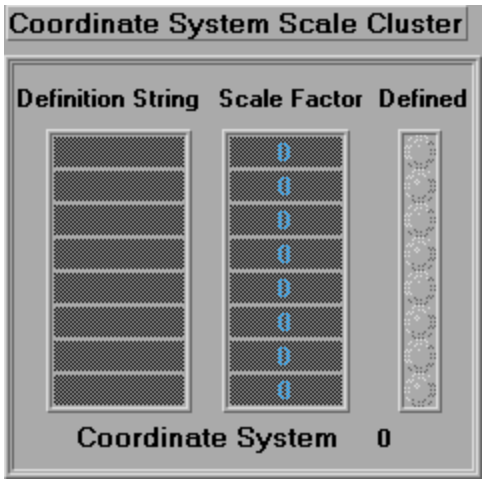
is generated. If the distance error is greater than this limit, a run-time error will be generated, and the program will stop
Examples
Given the program segment
INC CIRCLE1 F2
X7.072 Y7.072 R5
technically no circular arc path can be found, because the distance is SQRT(7.0722+7.0722) = 10.003, which is greater than twice the radius of 5. However as long as Ix96 is greater than 0.003, PMAC will create a near-circular path to the end point.

ix91: Default Program Number This parameter tells PMAC which motion program to run in this coordinate system when commanded to run from the control-panel input (START/ or STEP/ line taken low). It performs the same function for a hardware "run" command as the B command does for a software "run" command (R). It is intended primarily for stand-alone PMAC applications. The first use of a B command from a host computer for this coordinate system overrides this parameter.

ix92: Move Blend Disable If this parameter set to 0, programmed blended moves -- LINEAR- and CIRCLE-mode -- are blended together with no intervening stop. Upcoming moves are calculated during the current moves. If this parameter is set to 1, there is a brief stop in between each programmed move (it effectively adds a DWELL 0 command), during which the next move is calculated. The calculation time for the next move is determined by I11.

PmacCoordScale

This cluster contains three arrays of information about motor definitions in the addressed CS. The arrays are Definition String, Scale Factor, and Defined.



Coordinate System Scale Cluster Cluster containing information about the motor definitions in the addressed CS.

Definition String Motor definition string in the addressed CS.

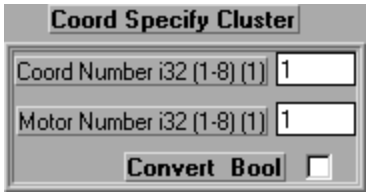
Scale Factor Array of encoder to CS Unit scale factors for each motor. 1.0 if the motor is not defined in the addressed CS.

Defined Array of Booleans. Element is TRUE if the motor is defined in the CS.

Coordinate System Addressed CS.

PmacCoordSpecify

Cluster defining the motor, CS, and conversion state to be applied



Coord Specify Cluster Cluster defining the motor, CS, and conversion state to be applied

Coord Number i32 (1-8) (1) Coordinate number to use

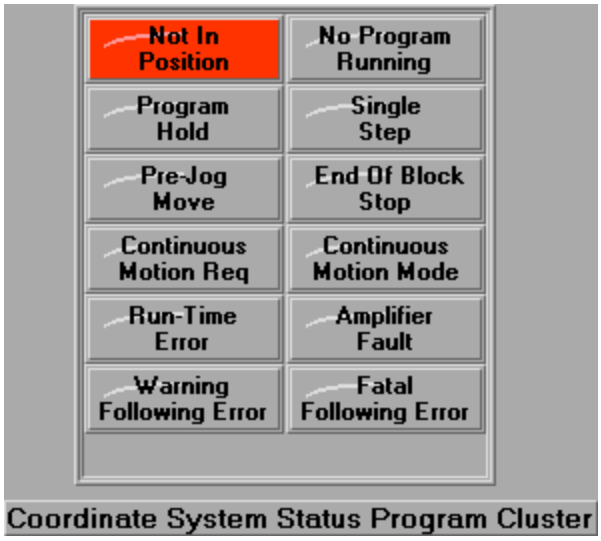
Motor Number i32 (1-8) (1) Motor Number to use

Convert Bool Apply a conversion for the specified motor in the specified CS

PmacCoordStatProg

This is an indicator cluster for the most common bits of CS status. It contains a label to display which CS the status is for.

The colors of the indicators are indicative of the information conveyed by the particular bit. For example 'Run-Time Error' is Red when TRUE and gray otherwise.



Coordinate System Status Program Cluster This is an indicator cluster for the most common coordinate system program status bits

In Position Status Bit 17 Word 2 - In Position: Green when all motors in the coordinate system are "in position". Three conditions must apply for this to be true: the desired velocity must be zero for all motors, the coordinate system cannot be in any timed move (even zero distance) or DWELL, and all motors must have a following error smaller than their respective Ix28 in-position bands. Red otherwise.

Program Running Status Bit 0 Word 1- Running Program: Green if the coordinate system is currently executing a motion program. Red if the C.S. is not currently executing a motion program. Note that it becomes Red as soon as it has calculated the last move and reached the final RETURN statement in the program, even if the motors are still executing the last move or two that have been calculated. Compare to the motor Running Program status bit.

Program Hold Status Bit 23 Word 2 - Gray when there is no Program Hold Stop Green when there is a Program Hold Stop.

Single Step Status Bit 1 Word 1 - Single-Step Mode: Green if the motion program currently executing in this coordinate system has been told to "step" one

move or block of moves, or if it has been given a Q (Quit) command. Gray if the motion program is executing a program by a R (run) command, or if it is not executing a motion program at all.

Pre-Jog Move Status Bit 8 Word 2 - Pre-jog Move Flag: Green when any motor in the coordinate system is executing a jog move to "pre-jog" position (J= command). Gray otherwise.

End of Block Stop Status Bit 14 Word 2 - End of Block Stop: Green when a motion program running in the currently addressed Coordinate System is stopped using the '/' command from a segmented move (Linear or Circular mode with I13 > 0). Gray otherwise.

Continuous Motion Req Status Bit 2 Word 1 - Continuous Motion Request: Green if the coordinate system has requested of it a continuous set of moves (e.g. with an R command). Gray if this is not the case (e.g. not running program, Ix92=1, or running under an S command).

Continuous Motion Mode Status Bit 2 Word 1 - Continuous Motion Mode: Green if the coordinate system is in a sequence of moves that it is blending together without stops in between. Gray if it is not currently in such a sequence, for whatever reason.

Run-Time Error Status Bit 22 Word 2 - Run-Time Error: Green when the coordinate system has stopped a motion program due to an error encountered while executing the program (e.g. jump to non-existent label, insufficient calculation time, etc.). Gray otherwise.

Amplifier Fault Status Bit 20 Word 2 - Amplifier Fault Error: Red when any motor in the coordinate system has been killed due to receiving an amplifier fault signal. Green at other times, changing from Red to Gray when the offending motor is re-enabled.

Warning Following Error Status Bit 18 Word 2 - Warning Following Error: Red when any motor in the coordinate system has exceeded its warning following error limit (Ix12). It stays Red if a motor has been killed due to fatal following error limit. Grey at all other times. The change from Red to Grey occurs when the offending motor's following error is reduced to under the limit, or if killed on fatal following error as well, when it is re-enabled.

Fatal Following Error Status Bit 19 Word 2 - Fatal Following Error: Red when any motor in the coordinate system has been killed due to exceeding

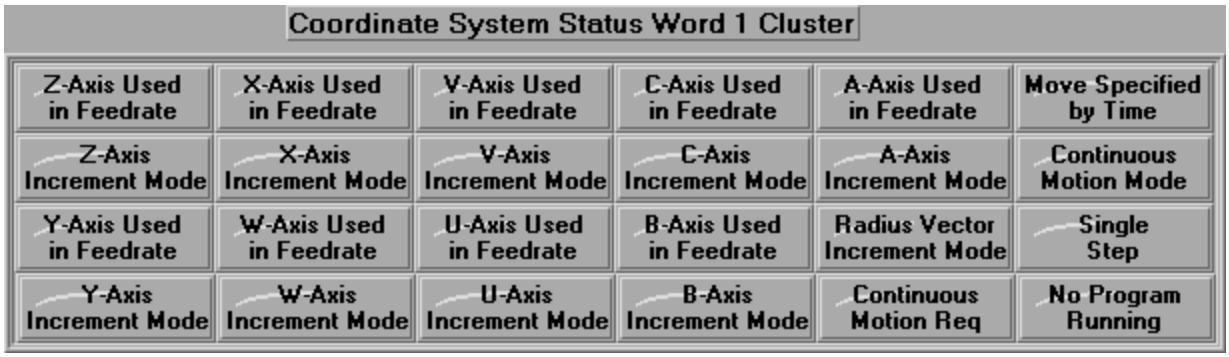
its fatal following error limit (Ix11). Grey at other times. The change from Red to Green occurs when the offending motor is re-enabled.

C.S. Description of currently addressed Coordinate System.

PmacCoordStatWord1

This is an indicator cluster for the first status word of a CS

The colors of the indicators are indicative of the information conveyed by the particular bit. For example 'Program Running' is Green when TRUE and grey otherwise.



Coordinate System Status Word 1 Cluster This is an indicator cluster for the first status word of a CS

Program Running Status Bit 0 Word 1 - Running Program: This bit is 1 if the coordinate system is currently executing a motion program. It is 0 if the C.S. is not currently executing a motion program. Note that it becomes 0 as soon as it has calculated the last move and reached the final RETURN statement in the program, even if the motors are still executing the last move or two that have been calculated. Compare to the motor Running Program status bit.

Single Step Status Bit 1 Word 1 - Single-Step Mode: This bit is 1 if the motion program currently executing in this coordinate system has been told to "step" one move or block of moves, or if it has been given a Q (Quit) command. It is 0 if the motion program is executing a program by a R (run) command, or if it is not executing a motion program at all.

Continuous Motion Mode Status Bit 2 Word 1 - Continuous Motion Mode: This bit is 1 if the coordinate system is in a sequence of moves that it is blending together without stops in between. It is 0 if it is not currently in such a sequence, for whatever reason.

Move Specified By Time Status Bit 3 Word 1 - Move-Specified-by-Time Mode: This bit is 1 if programmed moves in this coordinate system are currently specified by time (TM or TA), and the move speed is derived. It is 0 if programmed moves in this coordinate system are currently specified by feedrate (speed; F) and the move time is derived.

Continuous Motion Req Status Bit 4 Word 1 - Continuous Motion Request: This bit is 1 if the coordinate system has requested of it a continuous set of moves (e.g. with an R command). It is 0 if this is not the case (e.g. not running program, Ix92=1, or running under an S command).

Radius Vector Increment Mode Status Bit 5 Word 1 - Radius Vector Incremental Mode: This bit is 1 if circle move radius vectors are specified incrementally (i.e. from the move start point to the arc center). It is 0 if circle move radius vectors are specified absolutely (i.e. from the XYZ origin to the arc center). See the INC(R) and ABS(R) commands.

A-Axis Incremental Mode Status Bit 6 Word 1 - A-Axis Incremental Mode: This bit is 1 if this axis is in incremental mode -- moves specified by distance from the last programmed point. It is 0 if this axis is in absolute mode -- moves specified by end position, not distance. See the INC and ABS commands.

A-Axis Used in Feedrate Status Bit 7 Word 1 - A-Axis Used in Feedrate Calculations: This bit is 1 if this axis is used in the vector feedrate calculations for F-based moves in the coordinate system; it is 0 if this axis is not used. See the FRAX command.

B-Axis Incremental Mode Status Bit 8 Word 1 - B-Axis Incremental Mode: This bit is 1 if this axis is in incremental mode -- moves specified by distance from the last programmed point. It is 0 if this axis is in absolute mode -- moves specified by end position, not distance. See the INC and ABS commands.

B-Axis Used in Feedrate Status Bit 9 Word 1 - B-Axis Used in Feedrate Calculations: This bit is 1 if this axis is used in the vector feedrate calculations for F-based moves in the coordinate system; it is 0 if this axis is not used. See the FRAX command.

C-Axis Incremental Mode Status Bit 10 Word 1 - C-Axis Incremental Mode: This bit is 1 if this axis is in incremental mode -- moves specified by distance from the last programmed point. It is 0 if this axis is in absolute mode -- moves specified by end position, not distance. See the INC and ABS commands.

C-Axis Used in Feedrate Status Bit 11 Word 1 - C-Axis Used in Feedrate Calculations: This bit is 1 if this axis is used in the vector feedrate calculations for F-based moves in the coordinate system; it is 0 if this axis is not used. See the FRAX command.

U-Axis Incremental Mode Status Bit 12 Word 1 - U-Axis Incremental Mode: This bit is 1 if this axis is in incremental mode -- moves specified by distance from the last programmed point. It is 0 if this axis is in absolute mode -- moves specified by end position, not distance. See the INC and ABS commands.

U-Axis Used in Feedrate Status Bit 13 Word 1 - U-Axis Used in Feedrate Calculations: This bit is 1 if this axis is used in the vector feedrate calculations for F-based moves in the coordinate system; it is 0 if this axis is not used. See the FRAX command.

V-Axis Incremental Mode Status Bit 14 Word 1 - V-Axis Incremental Mode: This bit is 1 if this axis is in incremental mode -- moves specified by distance from the last programmed point. It is 0 if this axis is in absolute mode -- moves specified by end position, not distance. See the INC and ABS commands.

V-Axis Used in Feedrate Status Bit 15 Word 1 - V-Axis Used in Feedrate Calculations: This bit is 1 if this axis is used in the vector feedrate calculations for F-based moves in the coordinate system; it is 0 if this axis is not used. See the FRAX command.

W-Axis Incremental Mode Status Bit 16 Word 1 - W-Axis Incremental Mode: This bit is 1 if this axis is in incremental mode -- moves specified by distance from the last programmed point. It is 0 if this axis is in absolute mode -- moves specified by end position, not distance. See the INC and ABS commands.

W-Axis Used in Feedrate Status Bit 17 Word 1 - W-Axis Used in Feedrate Calculations: This bit is 1 if this axis is used in the vector feedrate calculations for F-based moves in the coordinate system; it is 0 if this axis is not used. See the FRAX command.

X-Axis Incremental Mode Status Bit 18 Word 1 - X-Axis Incremental Mode: This bit is 1 if this axis is in incremental mode -- moves specified by distance from the last programmed point. It is 0 if this axis is in absolute mode -- moves specified by end position, not distance. See the INC and ABS commands.

X-Axis Used in Feedrate Status Bit 19 Word 1 - X-Axis Used in Feedrate Calculations: This bit is 1 if this axis is used in the vector feedrate calculations for

F-based moves in the coordinate system; it is 0 if this axis is not used. See the FRAX command.

Y-Axis Incremental Mode Status Bit 20 Word 1 - Y-Axis Incremental Mode: This bit is 1 if this axis is in incremental mode -- moves specified by distance from the last programmed point. It is 0 if this axis is in absolute mode -- moves specified by end position, not distance. See the INC and ABS commands.

Y-Axis Used in Feedrate Status Bit 21 Word 1 - Y-Axis Used in Feedrate Calculations: This bit is 1 if this axis is used in the vector feedrate calculations for F-based moves in the coordinate system; it is 0 if this axis is not used. See the FRAX command.

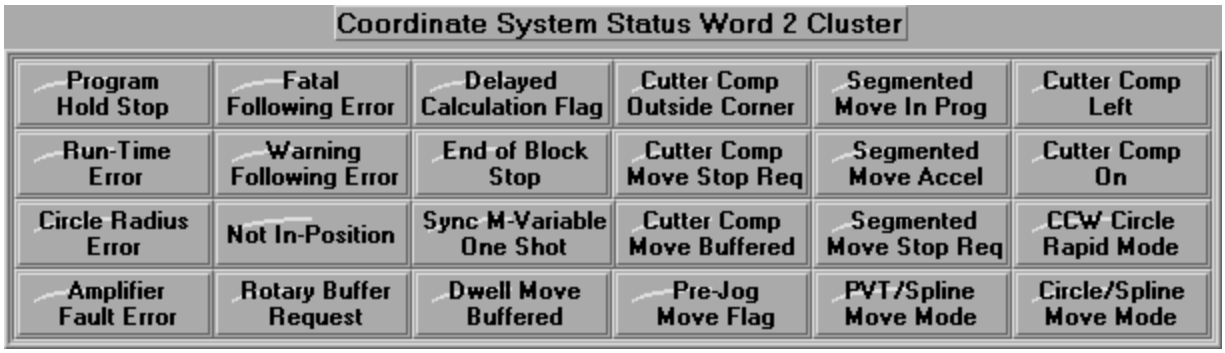
Z-Axis Incremental Mode Status Bit 22 Word 1 - Z-Axis Incremental Mode: This bit is 1 if this axis is in incremental mode -- moves specified by distance from the last programmed point. It is 0 if this axis is in absolute mode -- moves specified by end position, not distance. See the INC and ABS commands.

Z-Axis Used in Feedrate Status Bit 23 Word 1 - Z-Axis Used in Feedrate Calculations: This bit is 1 if this axis is used in the vector feedrate calculations for F-based moves in the coordinate system; it is 0 if this axis is not used. See the FRAX command.

PmacCoordStatWord2

This is an indicator cluster for the second status word of a CS

The colors of the indicators are indicative of the information conveyed by the particular bit. For example 'Run-Time Error' is Red when TRUE and grey otherwise.



Coordinate System Status Word 2 Cluster This is an indicator cluster for the second status word of a CS

Circle/Spline Move Mode Status Bit 0 Word 2 - CIRCLE/SPLINE Move Mode: This bit is 1 if the coordinate system is in either CIRCLE or SPLINE move mode. (If bit 4 of this word is 0, this means

CIRCLE mode; if bit 4 is 1, this means SPLINE mode.) This bit is 0 if the coordinate system is in a different move mode (LINEAR, PVT, or RAPID.)

CCW Circle/Rapid Mode Status Bit 1 Word 2 - CCW Circle Mode: This bit is 1 if the coordinate system is in CIRCLE2 (counterclockwise arc) move mode. It is 0 if the coordinate system is not in a CIRCLE mode, or if it is in CIRCLE1 mode (clockwise arc).

Cutter Comp On Status Bit 2 Word 2 - Cutter Compensation On: This bit is 1 if the coordinate system has cutter compensation on. It is 0 if cutter compensation if off.

Cutter Comp Left Status Bit 3 Word 2 - Cutter Compensation Left: This bit is 1 if the coordinate system has cutter compensation on, and the compensation is to the left when looking in the direction of motion. It is 0 if compensation is to the right, or if cutter compensation is off.

PVT/Spline Move Mode Status Bit 4 Word 2 - PVT/SPLINE Move Mode: This bit is 1 if this coordinate system is in either PVT move mode or SPLINE move mode. (If bit 0 of this word is 0, this means PVT mode; if bit 0 is 1, this means SPLINE mode.) This bit is 0 if the coordinate system is in a different move mode (LINEAR, CIRCLE, or RAPID).

Segmented Move Stop Request Status Bit 5 Word 2 - Segmented Move Stop Request: This bit is 1 when the coordinate system is executing motion program move in segmentation mode (I13>0) and it is decelerating to a stop. It is 0 otherwise. This is primarily for internal use.

Segmented Move Acceleration Status Bit 6 Word 2 - Segmented Move Acceleration: This bit is 1 when the coordinate system is executing motion program moves in segmentation mode (I13>0) and accelerating from a stop. It is 0 otherwise. This is primarily for internal use.

Segmented Move In Progress Status Bit 7 Word 2 - Segmented Move in Progress: This bit is 1 when the coordinate system is executing motion program moves in segmentation mode (I13>0). It is 0 otherwise. This is primarily for internal use.

Pre-Jog Move Flag Status Bit 8 Word 2 - Pre-jog Move Flag: This bit is 1 when any motor in the coordinate system is executing a jog move to "pre-jog" position (J= command). It is 0 otherwise.

Cutter Comp Move Buffered Status Bit 9 Word 2 - Cutter Comp Move Buffered: This bit is 1 when the coordinate system is executing moves with cutter compensation enabled, and the next move has been calculated and buffered. This is primarily for internal use.

Cutter Comp Move Stop Request Status Bit 10 Word 2 - Cutter Comp Move Stop Request: This bit is 1 when the coordinate system is executing moves with cutter compensation enabled, and has been asked to stop move execution. This is primarily for internal use.

Cutter Comp Outside Corner

Dwell Move Buffered

Sync M-Variable One-Shot

End of Block Stop

Delayed Calculation

Rotary Buffer Request Status Bit 16 Word 2 - Rotary Buffer Full: This bit is 1 when a rotary buffer exists for the coordinate system and enough program lines have been sent to it so that the buffer contains more than I16 lines ahead of what has been calculated.

In-Position Status Bit 17 Word 2 - In Position: This bit is 1 when all motors in the coordinate system are "in position". Three conditions must apply for this to be true: the desired velocity must be zero for all motors, the coordinate system cannot be in any timed move (even zero distance) or DWELL, and all motors must have a following error smaller than their respective Ix28 in-position bands.

Warning Following Error Status Bit 18 Word 2 - Warning Following Error: This bit is 1 when any motor in the coordinate system has exceeded its warning following error limit (Ix12). It stays at 1 if a motor has been killed due to fatal following error limit. It is 0 at all other times. The change from 1 to 0 occurs when the offending motor's following error is reduced to under the limit, or if killed on fatal following error as well, when it is re-enabled.

Fatal Following Error Status Bit 19 Word 2 - Fatal Following Error: This bit is 1 when any motor in the coordinate system has been killed due to exceeding its fatal following error limit (Ix11). It is 0 at other times. The change from 1 to 0 occurs when the

offending motor is re-enabled.

Amplifier Fault Error Status Bit 20 Word 2 - Amplifier Fault Error: This bit is 1 when any motor in the coordinate system has been killed due to receiving an amplifier fault signal. It is 0 at other times, changing from 1 to 0 when the offending motor is re-enabled.

Circle Radius Error Status Bit 21 Word 2 - Circle Radius Error: This bit is 1 when a motion program has been stopped because it was asked to do an arc move whose distance was more than twice the radius (by an amount greater than Ix97).

Run-Time Error Status Bit 22 Word 2 - Run-Time Error: This bit is 1 when the coordinate system has stopped a motion program due to an error encountered while executing the program (e.g. jump to non-existent label, insufficient calculation time, etc.)

Program Hold Stop Status Bit 23 Word 2 - Program Trace Active: This bit is 1 when the motion program trace feature is executing (after a TRACE command). It is 0 when the feature is not executing. The change from 1 to 0 occurs when an ENDTRACE command is received, or when the trace buffer fills up.

PmacDAQ

The VIs in this collection are PMACPanel examples or slightly modified versions of standard LabVIEW examples. See Chapter 9 for details.

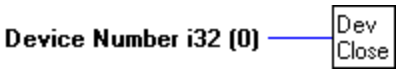
PmacDevice

\PmacDevice - Configure and manage access to PMAC using PComm32. Configuration of this access must match the driver configuration set using the control panel applet or MotionExe application.

VIs

PmacDevClose

Close the PComm32 device driver. PMAC will continue running as programmed as long as power is applied.



Device Number i32 (0) Provided by PmacDevOpen

PmacDevOpen

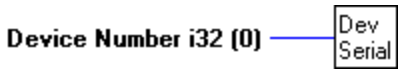
Open communication to PMAC using the PComm32 device driver. Check type, ROM date, and ROM Version. Provide Device Number for other VI's. You can select the mode of communication using the Communication Mode drop down menu. To make the selection permanent make your selection the default use the right mouse button and "Data Operations » Make Current Value Default" option. This SHOULD be done in conjunction with the options available on the PMAC control panel.

If PMACPanel communication doesn't appear to be working correctly see the manual. Specifically, check the section on PMAC Communication I-Variables.

I32 **Device Number i32 (0)** ID to provide access to PComm32 and PMAC.

PmacDevSerial

Allows the configuration of serial port communication. This SHOULD really be done in conjunction with PmacDevOpen and the PMAC control panel.



I32 **Device Number i32 (0)** Provided by PmacDevOpen

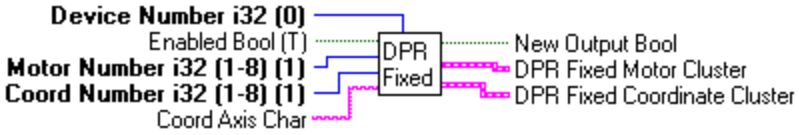
PmacDPR

\PmacDPR – A large collection of VIs for configuring and operating the many modes of DPR supported by PMAC. There are numerous examples demo nstrating the use of DPR and how to modify the supplied collection to suit your purposes.

VIs

PmacDPRFixedBack

Once DPR Fixed Background buffer operation is enabled this VI can be used to fetch the data for a specific Motor Number and Coord Number. The input Enabled can be used to enable and disable the actual fetch. The Default, un-wired, condition is TRUE. Coord Axis Char is a string (X, Y, Z, A, B, C, U, V, W) indicating which axis in Coord Number Comm'd Pos will represent. When New Output is TRUE DPR Fixed Motor Cluster and DPR Fixed Coordinate Cluster contain the most recent background data. When Enabled is FALSE the two output clusters contain the last valid data even though New Output is FALSE.



Device Number i32 (0) Device Number

Motor Number i32 (1-8) (1) Motor Number to fetch DPR Fixed Background data for

Enabled Bool (T) When TRUE query PMAC for the DPR Fixed Background buffer data for Motor Number and Coord Number. The VI maintains the enabled state. When FALSE do not place the query. Default is TRUE.

Coord Number i32 (1-8) (1) Coordinate System Number to fetch DPR Fixed Background data for

Coord Axis Char Single character string (X, Y, Z, A, B, C, U, V, W) defining which axis within Coord Number to use when presenting Comm'd Pos in DPR Fixed Coordinate Cluster.

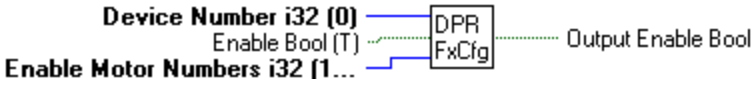
New Output Bool TRUE when DPR Fixed Motor Cluster and DPR Fixed Coordinate Cluster contain new data.

DPR Fixed Motor Cluster

DPR Fixed Coordinate Cluster

PmacDPRFixedBackConfig

DPR Fixed Background buffer is configured to update motor information for all motor between 1 and Enable Motor Numbers when Enable is TRUE. Operation is disabled when Enable is FALSE. Output Enable is TRUE when operation is enabled. The state is maintained by the VI. Operation of DPR Fixed Background buffers overlaps with DPR Real Time operation in that the number of motors enabled must be the same.



Device Number i32 (0) Device Number

Enable Bool (T) When TRUE configure the DPR Fixed Background buffer to fetch the number of motors specified by Enable Motor Numbers. The VI maintains the enabled state. When FALSE disable all background operations including Variable Background operations. Default is TRUE.

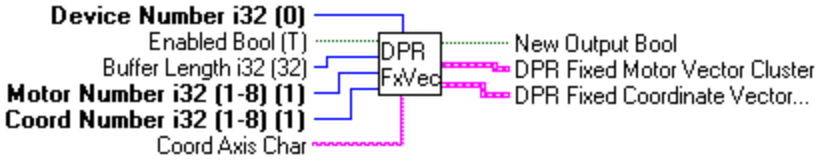
Enable Motor Numbers i32 (1-8) (1) The number of motors for DPR Fixed Background operations.

Output Enable Bool TRUE when DPR Fixed Background

operation is enabled. The output state is maintained by the VI.

PmacDPRFixedBackVectors

Once DPR Fixed Background buffer operation is enabled this VI can be used to fetch the data for a specific Motor Number and Coord Number. The input Enabled can be used to enable and disable the actual fetch. The Default, un-wired, condition is TRUE. Coord Axis Char is a string (X, Y, Z, A, B, C, U, V, W) indicating which axis in Coord Number Comm'd Pos will represent. When New Output is TRUE DPR Fixed Motor Cluster and DPR Fixed Coordinate Cluster contain the most recent background data. When Enabled is FALSE the two output clusters contain the last valid data even though New Output is FALSE.



Device Number i32 (0) Device Number

Motor Number i32 (1-8) (1) Motor Number to fetch DPR Fixed Background data for

Enabled Bool (T) When TRUE query PMAC for the DPR Fixed Background buffer data for Motor Number and Coord Number. The VI maintains the enabled state. When FALSE do not place the query. Default is TRUE.

Coord Number i32 (1-8) (1) Coordinate System Number to fetch DPR Fixed Background data for

Coord Axis Char Single character string (X, Y, Z, A, B, C, U, V, W) defining which axis within Coord Number to use when presenting Comm'd Pos in DPR Fixed Coordinate Vector Cluster.

Buffer Length i32 (32) Number of DPR Fixed Background samples to buffer when building vectors. The Default value is 32.

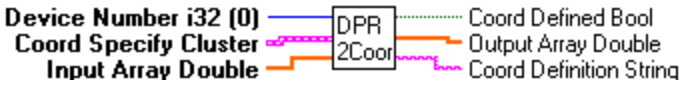
New Output Bool TRUE when DPR Fixed Motor Vector Cluster and DPR Fixed Coordinate Vector Cluster contain new data.

DPR Fixed Motor Vector Cluster Cluster of vectors containing DPR Fixed Background buffer data for the specified motor number.

DPR Fixed Coordinate Vector Cluster Cluster of vectors containing DPR Fixed Background buffer data for the specified coordinate system.

PmacDPRMotorVecToCoord

Coord Specify Cluster specifies a motor within a CS and an attempt to convert Input Array from encoder counts to CS units. If the motor is not defined in the CS no conversion is applied. If the motor is defined and Convert is TRUE Coord Defined is TRUE and Output Array is scaled from encoder counts to CS units. Coord Definition is a string specifying Output Value units as "Encoder" or the CS definition of the motor.



Input Array Double Array of values provided by PmacDPRRealTimeVectors, PmacDPRFixedVectors, or PmacDPRVarBackVectors that is to be converted to coodinate system units.

Value Motor position, velocity, of following error from PmacMotor VIs to possibly convert to CS units

Device Number i32 (0) Provided by PmacDevOpen

Coord Specify Cluster Cluster defining the motor, CS, and conversion state to be applied

Output Array Double Array of values in motor encoder units or coordinate system units.

Value Motor position, velocity, of following error from PmacMotor VIs to possibly convert to CS units

Coord Defined Bool If Motor Number is defined in Coord Number and Convert Bool is TRUE this is TRUE.

Coord Definition String String defining the motor within the CS. "Encoder" if not defined.

PmacDPRNumericCINCluster

This VI is used to Set or Get a collection of M-Variable's defined in DPR. It serves as a base VI from which you can create your own custom VIs. To use this VI you should make a copy of it, customize the Input and Output Clusters, and modify the contents of the diagram.

This VI functions identically to PmacDPRNumericCluster but is implemented with a CIN Node. It is slightly faster but provides more control over how the data is processed if speed or flexibility is an issue.

When Set/Get is FALSE - default state - Response Available is TRUE and Output DPR Numeric Cluster contains the fetched data. When Set/Get is TRUE Output DPR Numeric Cluster = Input DPR Numeric Cluster, Response Available is FALSE and the specified DPR locations are set.

Input DPR Numeric Cluster Input cluster of data for writing to DPR M-Variables during Set operations.

Set\Get Bool (F) Set the DPR contents using Input DPR Numeric Cluster when TRUE. Get the data for Output DPR Numeric Cluster when FALSE or unwired.

Device Number i32 (0) Provided by PmacDevOpen

Output DPR Numeric Cluster Custom data cluster read from DPR M-Variables during Get operation.

Response Available Bool (F) TRUE when Get operation produces a valid Output DPR Numeric Cluster

PmacDPRNumericCluster

This VI is used to Set or Get a collection of M-Variable's defined in DPR. It serves as a base VI from which you can create your own custom VIs. To use this VI you should make a copy of it, customize the Input and Output Clusters, and modify the contents of the diagram.

When Set/Get is FALSE - default state - Response Available is TRUE and Output DPR Numeric Cluster contains the fetched data. When Set/Get is TRUE Output DPR Numeric Cluster = Input DPR Numeric Cluster, Response Available is FALSE and the specified DPR locations are set.

Input DPR Numeric Cluster Input cluster of data for writing to DPR M-Variables during Set operations.

Set\Get Bool (F) Set the DPR contents using Input DPR Numeric Cluster when TRUE. Get the data for Output DPR Numeric Cluster when FALSE or unwired.

Device Number i32 (0) Device Number

Output DPR Numeric Cluster Custom data cluster read from DPR M-Variables during Get operation.

Response Available Bool (F) TRUE when Get operation produces a valid Output DPR Numeric Cluster

PmacDPRNumericDbl

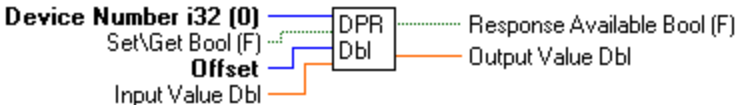
This VI is used to Set or Get PMAC double M-Variable's defined in DPR as M447->F:$DE03. PMAC handles the translation of PMAC's representation into Intel format when the F specification is used.

When Set/Get is FALSE - default state - Response Available is TRUE and Output Value is the value in DPR located at Offset. When Set/Get is TRUE Output Value = Input Value, Response Available is FALSE and the specified DPR location is set.

The mapping of PMAC addresses to PC addresses is involved and requires a bit of work to understand. In general, each 24 bit PMAC word requires one 32 bit PC word. For PMAC doubles specified as.

M447->F:$DE03

this offset should be $E03.



Device Number i32 (0) Device Number

Input Value Dbl Value to use for Set operation.

Set\Get Bool (F) Set the DPR Double at Offset using Input Value when TRUE. Get Output Value when FALSE or unwired.

Offset Offset of PMAC DPR M-Variable address.
The mapping of PMAC addresses to PC addresses is involved and requires a bit of work to understand. In general, each 24 bit PMAC word requires one 32 bit PC word. For PMAC doubles specified as.
M447->F:$DE03
this offset should be $E03.

Output Value Dbl Double representation of DPR value at Offset. If the response is not valid Output Value = Input Value.

Response Available Bool (F) TRUE when Get operation produces a valid Output Value

PmacDPRNumericDWord

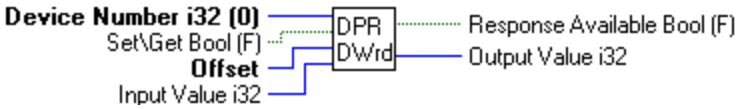
This VI is used to Set or Get PMAC long M-Variable's defined in DPR as M447->DP:$DE03. PMAC handles the translation of PMAC's representation into Intel format when the DP specification is used.

When Set/Get is FALSE - default state - Response Available is TRUE and Output Value is the value in DPR located at Offset. When Set/Get is TRUE Output Value = Input Value, Response Available is FALSE and the specified DPR location is set.

The mapping of PMAC addresses to PC addresses is involved and requires a bit of work to understand. In general, each 24 bit PMAC word requires one 32 bit PC word. For PMAC longs specified as.

M447->DP:$DE03

this offset should be $E03.



Device Number i32 (0) Device Number

Input Value i32 Value to use for Set operation.

Set\Get Bool (F) Set the DPR DWord (i32) at Offset using Input Value when TRUE. Get Output Value when FALSE or unwired.

Offset Offset of PMAC DPR M-Variable address.
The mapping of PMAC addresses to PC addresses is involved and requires a bit of work to understand. In general, each 24 bit PMAC word requires one 32 bit PC word. For PMAC longs specified as.
M447->DP:$DE03
this offset should be $E03.

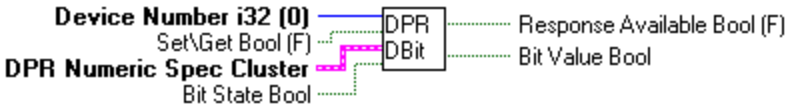
Output Value i32 i32 representation of DPR value at Offset.
If the response is not valid Output Value = Input Value.

Response Available Bool (F) TRUE when Get operation produces a valid Output Value

PmacDPRNumericDWordBit

This VI operates on the DPR DWord bit specified by DPR Numeric Spec Cluster.

When Set/Get is FALSE - the default state - the value of the bit is queried and returned by Bit Value with Response Available TRUE. When Set/Get is TRUE the specified bit is set to the value of Bit State - either TRUE or FALSE.



Device Number i32 (0) Device Number

Set\Get Bool (F) Set the specified bit using Bit State when TRUE. Get the bit state when FALSE or unwired.

Bit State Bool Bit State used during Set operation.

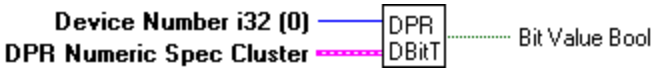
DPR Numeric Spec Cluster A cluster of items required to describe a DPR mapped PMAC M-Variable for bit and field access.

Response Available Bool (F) TRUE when Get operation produces a valid Bit Value

Bit Value Bool Value of specified bit during Get operations.
New value of specified bit as defined by Bit State during Set operations.

PmacDPRNumericDWordBitTest

This VI queries the DPR DWord bit specified by DPR Numeric Spec Cluster and returns the value in Bit Value.



Device Number i32 (0) Device Number

DPR Numeric Spec Cluster A cluster of items required to describe a DPR mapped PMAC M-Variable for bit and field access.

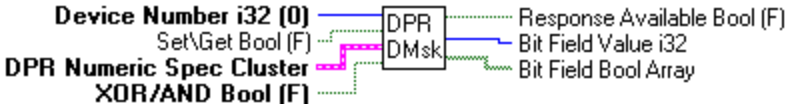
Bit Value Bool Value of specfiied bit for Boolean testing purposes.

PmacDPRNumericDWordSetMask

This VI operates on the DPR DWord bit field specified by DPR Numeric Spec Cluster.

When Set/Get is FALSE - the default state - the Mask specified by DPR Numeric Spec Cluster is AND'd with the specified address to produce the output Bit Field Value. Response Available is TRUE.

When Set/Get is TRUE the Mask is either OR'd or XOR'd with the contents of the field at the specified address. If XOR/OR is FALSE the mask is OR'd with the contents of the field at the specified address thereby setting bits specified by the mask. If XOR/OR is TRUE the mask is XOR'd with the contents of the field at the specified address thereby clearing the bits specified in the mask.



Device Number i32 (0) Device Number

Set\Get Bool (F) Set the specified bit field using the specified XOR/OR operation when TRUE. Get the bit field state and mask it using an AND operation when FALSE or unwired.

XOR/AND Bool (F) This defines the operation ro be applied to the contents of the specified bit field during Set/Get operations.
During Get operations the Mask specified by DPR Numeric Spec Cluster is AND'd with the specified address and bit field to produce the output Bit Field Value.
During Set operations the Mask is OR'd with the contents of the field at the specified address if this input is TRUE. If this input is FALSE the Mask is XOR'd with the contents of the field at the specified address. In both cases the output Bit Field is the result of the specified bit field AND the mask.

DPR Numeric Spec Cluster A cluster of items required to

describe a DPR mapped PMAC M-Variable for bit and field access.

Bit Field Value i32 Specified DPR bit field AND'd with the mask during Get operations.

Response Available Bool (F) TRUE when Get operation produces a valid Bit Field Value

Bit Field Bool Array A Boolean Array representation of the numerical value in Bit Field Value.

PmacDPRNumericSpec

This VI takes a DPR Numeric Spec Cluster defining a DPR address, variable type, and bit number of bit field and computes the numeric offset address of the item (Offset) and bit number of bit field (Bit Number).

The mapping of PMAC addresses to PC addresses is involved and requires a bit of work to understand. In general, each 24 bit PMAC word requires one 32 bit PC word. Hence, X:$DE03 and Y:$DE03 have different offsets in DPR.

For example, a PMAC Address such as:

M447->X:$DE03,8,1

M448->Y:$DE03,8,8

respectively have offsets of

$380E and $3810

in the PC.

DPR Numeric Spec Cluster A cluster of items required to describe a DPR mapped PMAC M-Variable for bit and field access.

Offset u32 The numeric offset in DPR as computed from the DPR Numeric Spec Cluster.

Bit Number u32 The bit number of field specifier for DPR bit and field operations.

PmacDPRNumericWord

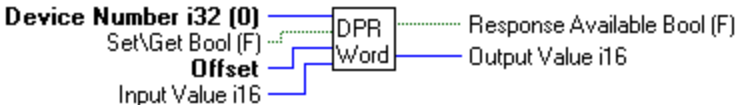
This VI is used to Set or Get PMAC long M-Variable's defined in DPR as M447->DP:$DE03 where the equivalent intel representation is 16 bits. PMAC handles the translation of PMAC's representation into Intel format when the DP specification is used.

When Set/Get is FALSE - default state - Response Available is TRUE and Output Value is the value in DPR located at Offset. When Set/Get is TRUE Output Value = Input Value, Response Available is FALSE and the specified DPR location is set.

The mapping of PMAC addresses to PC addresses is involved and requires a bit of work to understand. In general, each 24 bit PMAC word requires one 32 bit PC word. For PMAC longs specified as.

M447->DP:$DE03

this offset should be $E03.



Device Number i32 (0) Device Number

Input Value i16 Value to use for Set operation.

Set\Get Bool (F) Set the DPR Word (i16) at Offset using Input Value when TRUE. Get Output Value when FALSE or unwired.

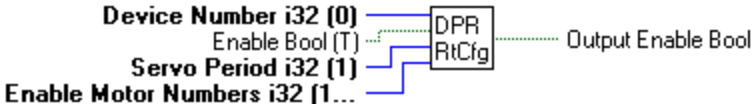
Offset Offset of PMAC DPR M-Variable address.
The mapping of PMAC addresses to PC addresses is involved and requires a bit of work to understand. In general, each 24 bit PMAC word requires one 32 bit PC word. For PMAC longs where the lower 16 bit are used specified as.
M447->DP:$DE03
this offset should be $E03.

Output Value i16 i16 representation of DPR value at Offset.
If the response is not valid Output Value = Input Value.

Response Available Bool (F) TRUE when Get operation produces a valid Output Value

PmacDPRRealTimeConfig

DPR Real Time Fixed buffer is configured to update motor information every Servo Period for all motor between 1 and Enable Motor Numbers when Enable is TRUE. Operation is disabled when Enable is FALSE. Output Enable is TRUE when operation is enabled. The state is maintained by the VI. Operation of DPR Real Time buffers overlaps with DPR Fixed Background operation in that the number of motors enabled must be the same.



Device Number i32 (0) Device Number

Enable Bool (T) When TRUE configure the DPR Fixed Real Time buffer to fetch the number of motors specified by Enable Motor Numbers. The VI maintains the enabled state. When FALSE disable all Real Time operations. Default is TRUE.

Enable Motor Numbers i32 (1-8) (1) The number of motors

for DPR Fixed Real Time operations.

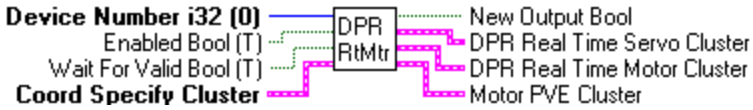
Servo Period i32 (1) Servo interval for sampling Real Time Motor servo data.

Output Enable Bool TRUE when DPR Fixed Real Time operation is enabled. The output state is maintained by the VI.

PmacDPRRealTimeMotor

Query PMAC DPR for the Real Time Fixed buffer Motor and Servo data. When Enabled is TRUE (the Default state) the data for Motor Number is fetched and used to build DPR Real Time Motor Cluster and DPR Real Time Servo Cluster. Motor PVE Cluster contains data in encoder counts or coordinate system units depending on the state of Coord Specify Cluster. See PmacMotorPVE for details on how this is done.

A successful query of PMAC's DPR depends on whether PMAC is accessing the memory. If Wait For Valid is TRUE the VI places queries to PMAC until a successful read at which time New Output is TRUE indicating valid output data. If Wait For Valid is FALSE the query may or may not succeed. If the query fails New Output is FALSE and the output clusters contain the data fetched during the last read.



Device Number i32 (0) Device Number

Enabled Bool (T) When TRUE query PMAC for the DPR Real Time buffer data for Motor Number. The VI maintains the enabled state. When FALSE do not place the query. Default is TRUE.

Coord Specify Cluster Cluster defining the motor, CS, and conversion state to be applied in the creation of Motor PVE Cluster

Wait For Valid Bool (T) When TRUE requery PMAC for DPR Real Time buffer data until the data is available. Otherwise fresh data is not returned and the outputs will contain the last available data.

New Output Bool TRUE when DPR Real Time Motor Cluster, DPR Real Time Servo Status Cluster, and Motor PVE Cluster contain new data. Otherwise the outputs contain the last available data.

DPR Real Time Motor Cluster Cluster containing motor servo values as sampled by the DPR Real Time facility.

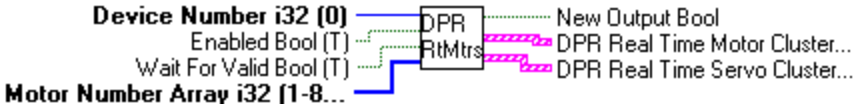
DPR Real Time Servo Cluster Cluster containing motor servo status as sampled by the DPR Real Time facility.

Motor PVE Cluster Cluster for PmacMotorPVE.ctl

PmacDPRRealTimeMotors

Query PMAC DPR Real Time Fixed buffer Motor and Servo data for the list of motors specified in Motor Number Array. When Enabled is TRUE (the Default state) the data for the specified motors is fetched and used to build an array of DPR Real Time Motor Clusters and DPR Real Time Servo Clusters.

A successful query of PMAC's DPR depends on whether PMAC is accessing the memory. If Wait For Valid is TRUE the VI places queries to PMAC until a successful read at which time New Output is TRUE indicating valid output data. If Wait For Valid is FALSE the query may or may not succeed. If the query fails New Output is FALSE and the output clusters contain the data fetched during the last read.



Device Number i32 (0) Device Number

Enabled Bool (T) When TRUE query PMAC for the DPR Real Time buffer data for motors specified in Motor Number Array. The VI maintains the enabled state. When FALSE do not place the query. Default is TRUE.

Motor Number Array i32 (1-8) (1 An array of Motor Numbers to fetch DPR Real Time buffer data for.

Wait For Valid Bool (T) When TRUE requery PMAC for DPR Real Time buffer data until the data is available. Otherwise fresh data is not returned and the outputs will contain the last available data.

New Output Bool TRUE when DPR Real Time Motor Cluster Array and DPR Real Time Servo Cluster Array contain new data. Otherwise the outputs contain the last available data.

DPR Real Time Motor Cluster Array An array of clusters containing motor servo values as sampled by the DPR Real Time facility.

DPR Real Time Motor Cluster Cluster containing motor servo values as sampled by the DPR Real Time facility.

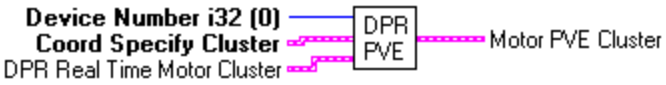
DPR Real Time Servo Cluster Array An array of clusters containing motor servo status as sampled by the DPR Real Time facility.

DPR Real Time Servo Cluster Cluster containing motor servo status as sampled by the DPR Real Time

facility.

PmacDPRRealTimePVE

Extract position, velocity, and following error from DPR Real TIme Motor Cluster assuming Motor Number operating in Coord Number. Assemble the measurements into Motor PVE Cluster. If Convert is TRUE convert the measurements to CS units. Otherwise leave them in encoder counts.



Device Number i32 (0) Provided by PmacDevOpen

Coord Specify Cluster Cluster defining the motor, CS, and conversion state to be applied

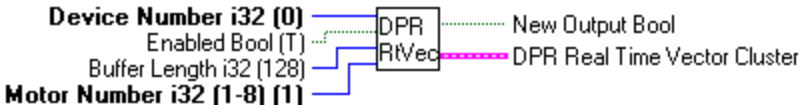
DPR Real Time Motor Cluster Cluster containing motor servo values as sampled by the DPR Real Time facility.

Motor PVE Cluster Cluster for PmacMotorPVE.ctl

PmacDPRRealTimeVectors

Query PMAC DPR for the Real Time Fixed buffer Motor and Servo data. When Enabled is TRUE (the Default state) the data for Motor Number is fetched and used to build DPR Real Time Motor Cluster and DPR Real Time Servo Cluster. Motor PVE Cluster contains data in encoder counts or coordinate system units depending on the state of Coord Specify Cluster. See PmacMotorPVE for details on how this is done.

A successful query of PMAC's DPR depends on whether PMAC is accessing the memory. If Wait For Valid is TRUE the VI places queries to PMAC until a successful read at which time New Output is TRUE indicating valid output data. If Wait For Valid is FALSE the query may or may not succeed. If the query fails New Output is FALSE and the output clusters contain the data fetched during the last read.



Device Number i32 (0) Device Number

Enabled Bool (T) When TRUE query PMAC for the DPR Real Time buffer data for Motor Number. The VI maintains the enabled state. When FALSE do not place the query. Default is TRUE.

Buffer Length i32 (128) Number of DPR Real Time samples to buffer when building vectors. The Default value is 128.

Motor Number i32 (1-8) (1) Motor Number to fetch the DPR

Real Time data for.

New Output Bool TRUE when DPR Real Time Vector Cluster contains new data. Otherwise the DPR Real Time Vector Cluster should NOT be used.

DPR Real Time Vector Cluster Cluster of vectors containing DPR Real Time buffer data for the specified motor number.

PmacDPRServoDelta

This diagnostic VI is used to take a time-sampled vector of servo timer values - or any other time sampled vector - and compute ST(k) - ST(k-1). Its primary purpose is to understand and analyze jitter in the sampling process.

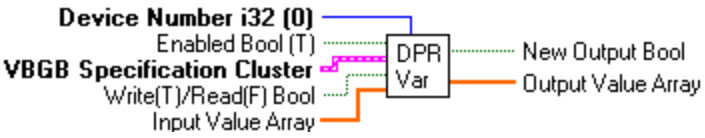
Servo Timer A vector of servo times gathered by PmacDPRRealTimeVectors or some other means.

Servo Timer Delta A vector of input Servo Timer sample differences computed by taking ST(k) - ST(k-1).

PmacDPRVarBack

Once DPR Variable Background buffer operation is enabled this VI can be used to fetch the data specified during the configuration. The input Enabled can be used to enable and disable the actual fetch. The Default, un-wired, condition is TRUE. When New Output is TRUE Output Value Array contains the most recent background data. When Enabled is FALSE Output Value Array contains the last valid data even though New Output is FALSE.

The Write/Read and Input Value Array inputs are not currently functional. Future releases may implement this capability.



Device Number i32 (0) Device Number

VBGB Specification Cluster Cluster containing information required to collect and convert DPR Variable Background items.

Input Value Array An array of doubles to be written using the DPR Variable Background buffer

Write(T)/Read(F) Bool The default case (F) reads the DPR Variable Background specified in the VBGB Specification

Cluster. The Write case (T) is available but not implemented.

Enabled Bool (T) When TRUE query PMAC for the DPR Variable Background buffer data configured by PmacDPRVarBackConfig and specified in VBGB Specification Cluster. The VI maintains the enabled state. When FALSE do not place the query. Default is TRUE.

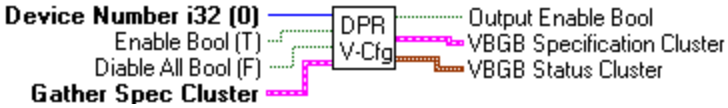
New Output Bool TRUE when DPR Output Value Array contains new data as the result of a read.

Output Value Array An array of doubles fetched from the DPR Variable Background buffer.

PmacDPRVarBackConfig

This VI creates a set of Address Items specified by Gather Spec Cluster using the DPR Variable Background when Enable is TRUE. The VI produces a VBGB Status Cluster with relevant information about this buffer and a VBGB Specification Cluster containging information required to actually fetch the data using PmacdPRVarBack and PmacDPRVarBackVectors.

Operation is disabled when Enable is FALSE. Output Enable is TRUE when operation is enabled. The state is maintained by the VI. This VI can be used multiple times to create sets of VBGB Address Items. See the documentation for limitations on how many sets can be created and their size.



Device Number i32 (0) Device Number

Enable Bool (T) When TRUE configure the DPR Variable Background buffer to fetch the items specified by Gather Spec Cluster. The VI maintains the enabled state. When FALSE disable VBGB specified. Default is TRUE.

Gather Spec Cluster Cluster containing servo sample rate and array of clusters of Address Items describing item, address, and scale factor.

Diable All Bool (F) When TRUE configure the DPR Variable Background buffer to fetch the items specified by Gather Spec Cluster. The VI maintains the enabled state. When FALSE disable VBGB specified. Default is TRUE.

Output Enable Bool TRUE when DPR Variable Background operation is enabled. The output state is maintained by the VI.

VBGB Status Cluster A collection of information about the DPR Variable Background buffer created by the execution of PmacDPRVarBackConfig. Most of this information is not of use to the general PMACPanel user.

VBGB Specification Cluster Cluster containing information required to collect and convert DPR Variable Background items.

PmacDPRVarBackgroundxLate

This VI converts an array of strings representing the gather addresses. For example:

DP:$0028

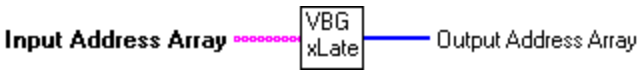
X:$0026

and converts them to an array of numeric addresses specifying DPR Variable Background items to collect.

PMAC addresses are specified using an array of long integers. The most significant word of each long (upper 16 bits) specifies the word type. A value of 0, 1, 2 and 4 corresponds to Y, Long, X, and SPECIAL respectively. For Y, Long and X entries the least significant word specifies the actual PMAC address to be copied.

Take for example the following define statements and the initialization of a long integer array.

```
#define MTR1_ACTULPOS       0x00010028 /* PMAC Address D:$0028 */
#define MTR1_ACTULVEL       0x00020033 /* PMAC Address X:$0033 */
#define MTR1_DACOUT         0x00020045 /* PMAC Address X:$0045 */
#define FB_10               0x0004000A /* PMAC function block #10 */
long myaddrarr[3]={MTR1_DACOUT,MTR1_ACTULPOS,MTR1_ACTULVEL};
```



Input Address Array An array of strings representing the addresss to gather. For example:
DP:$0028
X:$0026

Output Address Array Array of numeric addresses specifying DPR Variable Background items to collect. PMAC addresses are specified using an array of long integers. The most significant word of each long (upper 16 bits) specifies the word type. A value of 0, 1, 2 and 4 corresponds to Y, Long, X, and SPECIAL respectively. For Y, Long and X entries the least significant word specifies the actual PMAC address to be copied.
Take for example the following define statements and the initialization of a long integer array.

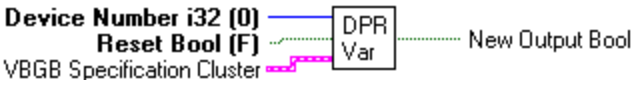
```
#define MTR1_ACTULPOS       0x00010028 /* PMAC
Address D:$0028 */
#define MTR1_ACTULVEL       0x00020033 /* PMAC
Address X:$0033 */
#define MTR1_DACOUT         0x00020045 /* PMAC
Address X:$0045 */
#define FB_10               0x0004000A /* PMAC
function block #10 */
```

long
myaddrarr[3]={MTR1_DACOUT,MTR1_ACTULPOS,MTR
1_ACTULVEL};

PmacDPRVarBackReset

Once DPR Variable Background buffer operation is enabled this VI can be used to fetch the data specified during the configuration. The input Enabled can be used to enable and disable the actual fetch. The Default, un-wired, condition is TRUE. When New Output is TRUE Output Value Array contains the most recent background data. When Enabled is FALSE Output Value Array contains the last valid data even though New Output is FALSE.

The Write/Read and Input Value Array inputs are not currently functional. Future releases may implement this capability.



Device Number i32 (0) Device Number

VBGB Specification Cluster Cluster containing information required to collect and convert DPR Variable Background items.

Reset Bool (F) When TRUE Stop all DPR Variable Background and Fixed Background operation.

New Output Bool TRUE when DPR Variable Background and Fixed Background operation is halted.

PmacDPRVarBackVectors

Query PMAC DPR for the Variable Background buffer data specified by VBGB Specification Cluster. When Enabled is TRUE (the Default state) the data is fetched and used to build Output Value 2D Array. When the number of samples equals Buffer Length New Output is TRUE and Output Value 2D Array contains the buffered data. If New Output is FALSE DO NOT use Output Value 2D Array.

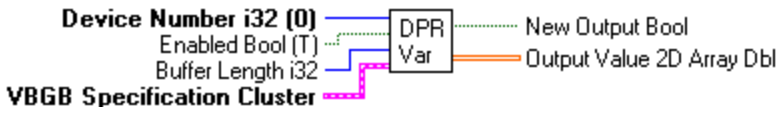
The implementation of this VI relies on pre-allocated buffers into which the gathered data is temporarily stored. The current buffers have the following limitations:

A maximum of 8 buffers. These are created by individual calls to PmacDPR VarBackConfig.

A maximum of 8 items per buffer.

A maximum of 128 samples per item.

See the documentation for information on changing these defaults.



Device Number i32 (0) Device Number

VBGB Specification Cluster Cluster containing information required to collect and convert DPR Variable Background items. Required by PmacDPRVarBack VIs.

Enabled Bool (T) When TRUE query PMAC for the DPR Variable Background buffer data specified by VBGB Specification Clusterr. The VI maintains the enabled state. When FALSE do not place the query. Default is TRUE.

Buffer Length i32 Number of DPR Variable Background samples to buffer when building vectors. The Default value is 32.

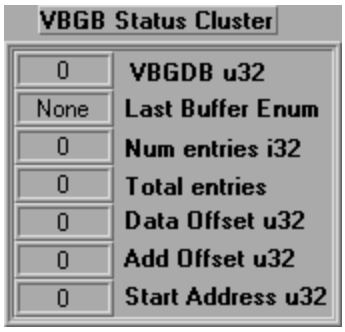
New Output Bool TRUE when Output Value 2D Array contains new data. Otherwise Output Value 2D Array should NOT be used.

Output Value 2D Array Dbl A two dimensional array containing the time sampled vectors of DPR Variable Background buffer data. The first index for each gathered item. The second index is the sample number for the specified item.

Indicator and Control Clusters

PmacDPRVarBackStat

A collection of information about the DPR Variable Background buffer created by the execution of PmacDPRVarBackConfig. Most of this information is not of use to the general PMACPanel user.



VBGB Status Cluster A collection of information about the DPR Variable Background buffer created by the execution of PmacDPRVarBackConfig. Most of this information is not of use to the general PMACPanel user.

VBGDB u32 The index of this DPR Variable Background buffer.

Last Buffer Enum This enumerated type identifies which buffer is the last variable sized buffer in DPR.

Use this type to decide which buffer (either a binary rotary or the variable background data buffer) can be initialized or removed. The sequence for the binary rotary and variable background data buffers is like a last in first out stack. The order of initialization must be: 1. Binary rotary buffer 0, 2. Binary rotary buffer 1, 3. Variable background data buffer.

Num entries i32 The number of items being updated in the buffer by the calling application.

Total entries The number of items being updated in the buffer by all applications. Should never exceed 128.

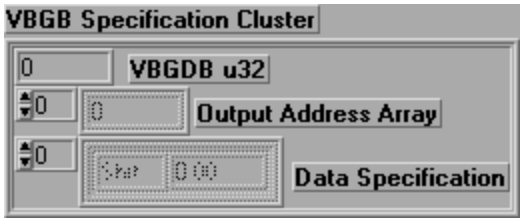
Data Offset u32 The offset, in bytes, from the beginning of DPR that the data for this VBGDB begins.

Add Offset u32 The offset, in bytes, from the beginning of DPR that the address array for this VBGDB begins. Divide by 4 to get the PMAC offset (from beginning of DPR 0xD000).

Start Address u32 The PMAC address of the start of the variable data buffer. This address also indicates the end of the free user memory.

PmacDPRVarBackSpec

Cluster containing information required to collect and convert DPR Variable Background items. Required by PmacDPRVarBack VIs.



VBGB Specification Cluster Cluster containing information required to collect and convert DPR Variable Background items. Required by PmacDPRVarBack VIs.

VBGDB u32 The index of this DPR Variable Background buffer.

Output Address Array Array of numeric addresses specifying DPR Variable Background items to collect.
PMAC addresses are specified using an array of long integers. The most significant word of each long (upper 16 bits) specifies the word type. A value of 0, 1, 2 and 4 corresponds to Y, Long, X,

and SPECIAL respectively. For Y, Long and X entries the least significant word specifies the actual PMAC address to be copied.
Take for example the following define statements and the initialization of a long integer array.

```
#define MTR1_ACTULPOS          0x00010028 /*
PMAC Address D:$0028 */
#define MTR1_ACTULVEL          0x00020033 /*
PMAC Address X:$0033 */
#define MTR1_DACOUT            0x00020045 /*
PMAC Address X:$0045 */
#define FB_10                  0x0004000A /*
PMAC function block #10 */
long
myaddrarr[3]={MTR1_DACOUT,MTR1_ACTULP
OS,MTR1_ACTULVEL};
```

Data Specification An array of VBGDG Spec Clusters for the configured collection of DPR Variable Background items.

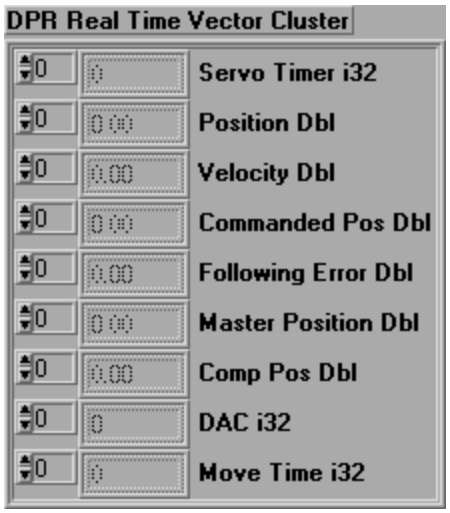
VBGDG Spec Cluster VBGDG Spec Cluster containing a scale factor and type for the specified item.

Scale Factor Scale Factor to apply to the data read from the DPR Variable Background buffer.

Address Item Type Enumerated type defining type of raw data

PmacDPRRealTimeVector

Cluster of vectors containing DPR Real Time buffer data for the specified motor number.

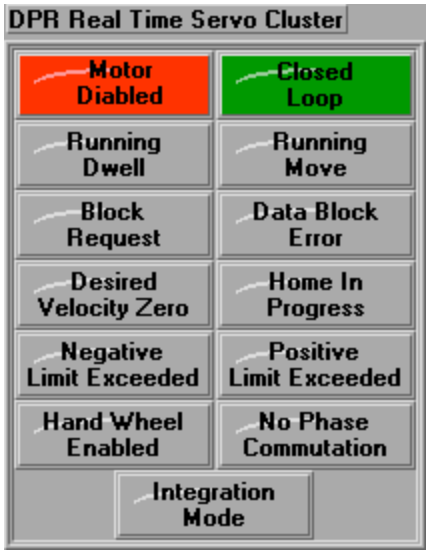


DPR Real Time Vector Cluster Cluster of vectors containing DPR Real Time buffer data for the specified motor number.

- **Servo Timer i32** PMAC's servo counter located at PMAC address 0.
- **Position Dbl** Actual motor position in encoder counts
- **Velocity Dbl** Motor velocity in units of 1/(Ix09 * 32) counts per servo cycle.
- **Commanded Pos Dbl** Commanded motor position in encoder counts.
- **Following Error Dbl** Following error in encoder counts
- **Master Position Dbl** Master position of the motor in encoder counts
- **Comp Pos Dbl** Compensation position in encoder counts.
- **DAC i32** Output DAC value of the previous servo cycle.
- **Move Time i32** Time in mS left in the currently executing move.

PmacDPRRealTimeServo

Cluster containing motor servo status as sampled by the DPR Real Time facility.



DPR Real Time Servo Cluster Cluster containing motor servo status as sampled by the DPR Real Time facility.

Home In Progress Status Bit 10 Word 1- Home Search in Progress: Green while the motor is searching for its home trigger -- it becomes Green as soon as the calculations for the move have started, and becomes Grey again as soon as it has found the trigger (which is before it has finished the entire move). This is not a good bit to observe to see if the homing move is complete. Use the Home Complete bit instead.

Block Request Status Bit 11 Word 1 - Block Request: This bit is 1 when the motor has just entered a new move section, and is requesting that the upcoming section be calculated. It is 0 otherwise. It is primarily for internal use.

Desired Velocity Zero Status Bit 13 Word 1 - Desired Velocity Zero: This bit is 1 if the motor is in closed-loop control and the commanded velocity is zero (i.e. it is trying to hold position). It is zero either if the motor is in closed-loop mode with non-zero commanded velocity, or if it is in open-loop mode.

Data Block Error Status Bit 14 Word 1 - Data Block Error: This bit is 1 when move execution has been aborted because the data for the next move section was not ready in time. This is due to insufficient calculation time. It is 0 otherwise. It changes from 1 to 0 when another move sequence is started.

Dwell In Progress Status Bit 15 Word 1 - Dwell in Progress: This bit is 1 when the motor's coordinate system is executing a DWELL instruction. It is 0 otherwise.

Integration Mode Status Bit 16 Word 1 - Integration Mode: This bit is 1 when Ix34 is 1 and the servo loop integrator is only active when desired velocity is zero. It is 0 when Ix34 is 0 and the servo loop integrator is always active.

Running Move Status Bit 17 Word 1 - Running Definite-Time Move: This bit is 1 when the motor is executing any move with a predefined end-point and end-time. This includes any motion program move dwell or delay, any jog-to-position move, and the portion of a homing search move after the trigger has been found. It is 0 otherwise. It changes from 1 to 0 when execution of the commanded move finishes.

Open Loop Status Bit 18 Word 1 - Open Loop Mode: Red when the servo loop for the motor is open, either with outputs enabled or disabled (killed). (Refer to Amplifier Enabled status bit to distinguish

between the two cases.) Green when the servo loop is closed (under position control, always with outputs enabled).

Phased Motor Status Bit 19 Word 1 - Phased Motor: This bit is 1 when Ix01 is 1 and this motor is being commutated by PMAC; it is 0 when Ix01 is 0 and this motor is not being commutated by PMAC.

Hand Wheel Enabled Status Bit 20 Word 1 - Handwheel Enabled: This bit is 1 when Ix06 is 1 and position following for this axis is enabled; it is 0 when Ix06 is 0 and position following is disabled.

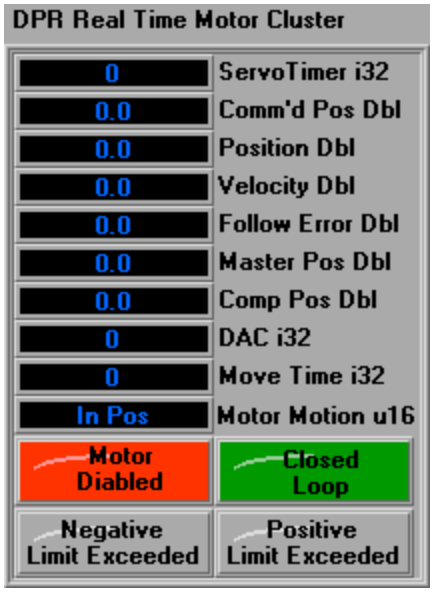
Neg Limit Exceeded Status Bit 22 Word 1- Negative End Limit Set: Red when motor actual position is less than the software negative position limit (Ix14), or when the hardware limit on this end (+LIMn -- note!) has been tripped; Grey otherwise.

Pos Limit Exceeded Status Bit 21 Word 1- Positive End Limit Set: Red when motor actual position is greater than the software positive position limit (Ix13), or when the hardware limit on this end (-LIMn -- note!) has been tripped; Grey otherwis e.

Motor Activated Status Bit 23 Word 1 - Motor Activated: Green when Ix00 is 1 and the motor calculations are active; Red when Ix00 is 0 and motor calculations are deactivated.

PmacDPRRealTimeMotor

Cluster containing motor servo values as sampled by the DPR Real Time facility.



DPR Real Time Motor Cluster Cluster containing motor servo values as sampled by the DPR Real Time facility.

ServoTimer i32 PMAC's servo counter located at PMAC address 0.

Comm'd Pos Dbl Commanded motor position in encoder counts.

Position Dbl Actual motor position in encoder counts

Velocity Dbl Motor velocity in units of 1/(Ix09 * 32) counts per servo cycle.

Follow Error Dbl Following error in encoder counts

Master Pos Dbl Master position of the motor in encoder counts

Comp Pos Dbl Compensation position in encoder counts.

DAC i32 Output DAC value of the previous servo cycle.

Move Time i32 Time in mS left in the currently executing move.

Motor Motion u16 Enumerated type defining the current motion state of the motor. The possibilities are:

In Position,
Jog
Running
Homing
Handle
Open Loop
Disabled

Motor Activated Status Bit 23 Word 1 - Motor Activated: Green when Ix00 is 1 and the motor calculations are active; Red when Ix00 is 0 and motor calculations are deactivated.

Open Loop Status Bit 18 Word 1 - Open Loop Mode: Red when the servo loop for the motor is open, either with outputs enabled or disabled (killed). (Refer to Amplifier Enabled status bit to distinguish between the two cases.) Green when the servo loop is closed (under position control, always with outputs enabled).

Neg Limit Exceeded Status Bit 22 Word 1- Negative End Limit Set: Red when motor actual position is less than the software negative position limit (Ix14),

or when the hardware limit on this end (+LIMn -- note!) has been tripped; Grey otherwise.

Pos Limit Exceeded Status Bit 21 Word 1- Positive End Limit Set: Red when motor actual position is greater than the software positive position limit (Ix13), or when the hardware limit on this end (-LIMn -- note!) has been tripped; Grey otherwise.

PmacDPRNumericSpec

A cluster of items required to describe a DPR mapped PMAC M-Variable for bit and field access.

DPR Numeric Spec Cluster A cluster of items required to describe a DPR mapped PMAC M-Variable for bit and field access.

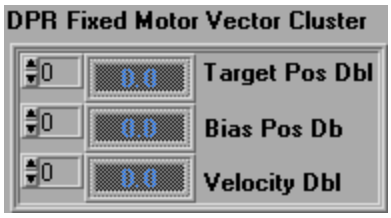
Address i32 Hexadecimal integer specifying DPR address offset. For example,
PMAC Addresses such as:
M445->F:$DE01
M446->DP:$DE02
M447->X:$DE03,8,1
Become
E01, E02, and E03 respectively.M446->DP:$DE02

X/Y String A single character string (X or Y) defining the type of data. Not for L or DP.

Mask/Bit i32 A hexadecimal value used to define a bit number for single bit operations or a multi digit hexadecimal number defining a mask for multi-bit operations.

PmacDPRFixedBackMotorVector

Cluster of vectors containing DPR Fixed Background buffer data for the specified motor number.



DPR Fixed Motor Vector Cluster Cluster of vectors containing DPR Fixed Background buffer data for the

specified motor number.

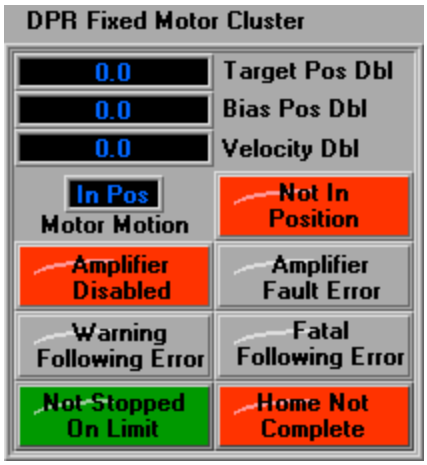
Target Pos Dbl Vector of Target positions for the motor in encoder counts.

Bias Pos Db Vector of Bias positions for the motor in encoder counts

Velocity Dbl Vector of Velocities for the motor in encoder counts per minute.

PmacDPRFixedBackMotor

Cluster for DPR Fixed Background buffer defining the state of a motor operating within a coordinate system.



DPR Fixed Motor Cluster Cluster for DPR Fixed Background buffer defining the state of a motor operating within a coordinate system.

Target Pos Dbl Target position of the motor in encoder counts.

Bias Pos Dbl Bias position of the motor in encoder counts

Velocity Dbl Velocity of the motor in encoder counts per minute.

Amplifier Enabled Status Bit 14 Word 2 - Amplifier Enabled: Green when the outputs for this motor's amplifier are enabled, either in open-loop or closed-loop mode (refer to Open-Loop Mode status bit to distinguish between the two cases). Red when the outputs are disabled (killed).

Amp Fault Status Bit 3 Word 2 - Amplifier Fault Error: Red if this motor has been disabled because of an amplifier fault error, even if the amplifier fault

signal has gone away. Grey at all other times, becoming Grey again when the motor is re-enabled.

Warning Following Error Status Bit 1 Word 2 - Warning Following Error: Red if the following error for the motor exceeds its warning following error limit (Ix12). It stays at Red if the motor is killed due to fatal following error. Grey at all other times, changing from Red to Grey when the motor's following error reduces to under the limit, or if killed, is re-enabled.

Fatal Following Error Status Bit 2 Word 2 - Fatal Following Error: Red if this motor has been disabled because it exceeded its fatal following error limit (Ix11). Grey at all other times, becoming Grey again when the motor is re-enabled.

Stopped on Limit Status Bit 11 Word 2 - Stopped on Position Limit: Red if this motor has stopped because of either a software or a hardware position (overtravel) limit, even if the condition that caused the stop has gone away. Green at all other times, even when into a limit but moving out of it.

Home Complete Status Bit 10 Word 2 - Home Complete: Red on power-up or reset, becomes Green when a homing search move is successfully completed. If a second homing move is done, this bit is set to Red at the beginning of the move, and only becomes Green again if that homing search move is successfully completed.

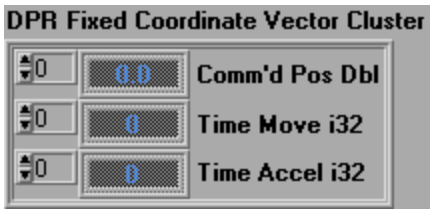
In Position Status Bit 0 Word 2 - In Position: Green when three conditions are satisfied: the desired velocity zero bit is 1 (which requires closed-loop control and no commanded move); the program timer is off (not currently executing any move, DWELL, or DELAY), and the magnitude of the following error is smaller than Ix28. Red otherwise.

Motor Motion Enumerated type defining the current motion state of the motor. The possibilities are:

In Position,
Jog
Running
Homing
Handle
Open Loop
Disabled

PmacDPRFixedBackCoordVector

Cluster of vectors containing DPR Fixed Background buffer data for the specified coordinate system.



DPR Fixed Coordinate Vector Cluster Cluster of vectors containing DPR Fixed Background buffer data for the specified coordinate system.

[DBL] **Comm'd Pos Dbl** Vector of Commanded motor positions for the specified axis in the specified coordinate system in the units specified for that axis.

[I32] **Time Move i32** Vector of Time in mS remaining in move for the specified coordinate system. It includes acceleration, steady state, and decleration times.

[I32] **Time Accel i32** Vector of Time in mS remaining in acceleration/deceleration when I13 > 0.

PmacDPRFixedBackCoord

Cluster for DPR Fixed Background buffer defining the state of a program executing within a coordinate system

DPR Fixed Coordinate Cluster Cluster for DPR Fixed Background buffer defining the state of a program executing within a coordinate system

Comm'd Pos Dbl Commanded motor position for the specified axis in the specified coordinate system in the units specified for that axis.

Time Move i32 Time in mS remaining in move for the specified coordinate system. It include acceleration, steady state, and decleration times.

Time Accel i32 Time in mS remaining in acceleration/deceleration when I13 > 0.

Prog Remain i32 Number of program lines remaining for the specified coordinate system. Same as 'PR' command.

Prog Exec i32 Current Program execution counter.

In Position Status Bit 0 Word 2 - In Position: Green when three conditions are satisfied: the desired velocity zero bit is 1 (which requires closed-loop control and no commanded move); the program timer is off (not currently executing any move, DWELL, or DELAY), and the magnitude of the following error is smaller than Ix28. Red otherwise.

Warning Following Error Status Bit 18 Word 2 - Warning Following Error: This bit is 1 when any motor in the coordinate system has exceeded its warning following error limit (Ix12). It stays at 1 if a motor has been killed due to fatal following error limit. It is 0 at all other times. The change from 1 to 0 occurs when the offending motor's following error is reduced to under the limit, or if killed on fatal following error as well, when it is re-enabled.

Fatal Following Error Status Bit 19 Word 2 - Fatal Following Error: This bit is 1 when any motor in the coordinate system has been killed due to exceeding its fatal following error limit (Ix11). It is 0 at other times. The change from 1 to 0 occurs when the offending motor is re-enabled.

Run-Time Error Status Bit 22 Word 2 - Run-Time Error: This bit is 1 when the coordinate system has stopped a motion program due to an error encountered while executing the program (e.g. jump to non-existent label, insufficient calculation time, etc.)

Circle Radius Error Status Bit 21 Word 2 - Circle Radius Error: This bit is 1 when a motion program has been stopped because it was asked to do an arc move whose distance was more than twice the radius (by an amount greater than Ix97).

Amplifier Fault Error Status Bit 20 Word 2 -

Amplifier Fault Error: This bit is 1 when any motor in the coordinate system has been killed due to receiving an amplifier fault signal. It is 0 at other times, changing from 1 to 0 when the offending motor is re-enabled.

Program Running Status Bit 0 Word 1 - Running Program: This bit is 1 if the coordinate system is currently executing a motion program. It is 0 if the C.S. is not currently executing a motion program. Note that it becomes 0 as soon as it has calculated the last move and reached the final RETURN statement in the program, even if the motors are still executing the last move or two that have been calculated. Compare to the motor Running Program status bit.

Single Step Status Bit 1 Word 1 - Single-Step Mode: This bit is 1 if the motion program currently executing in this coordinate system has been told to "step" one move or block of moves, or if it has been given a Q (Quit) command. It is 0 if the motion program is executing a program by a R (run) command, or if it is not executing a motion program at all.

Continuous Motion Mode Status Bit 2 Word 1 - Continuous Motion Mode: This bit is 1 if the coordinate system is in a sequence of moves that it is blending together without stops in between. It is 0 if it is not currently in such a sequence, for whatever reason.

Continuous Motion Req Status Bit 4 Word 1 - Continuous Motion Request: This bit is 1 if the coordinate system has requested of it a continuous set of moves (e.g. with an R command). It is 0 if this is not the case (e.g. not running program, Ix92=1, or running under an S command).

Motion Mode Enumerated type defining the current motion mode of the coordinate system. The possibilities are:

- Linear
- Rapid
- Circular CW
- Circular CCW
- Spline
- PVT

Prog Mode Enumerated type defining the current program mode of the coordinate system. The possibilities are:

- Stop
- Run
- Step
- Hold

Jog Hold
Jog Stop

PmacEncoder

\PmacEncoder – This collection provides VIs to access and control the encoder gate array for position capture and compare. There are also tools for developing background PLC programs for generating Compare-Equal outputs.

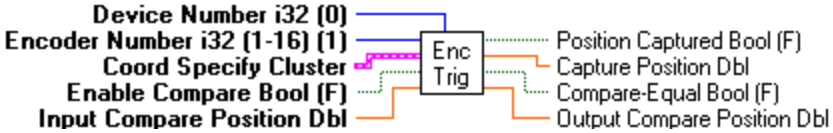
VIs

PmacEncoderTrigger

This VI maintains Encoder Number's compare-equal and capture operations and monitors the encoder's status register. Home offsets are removed or added during the processing of compare-equal and capture register data. Limitations associated with 24 bit rollowver are not handled by this VI.

When Enable Compare is TRUE Encoder Number's compare-equal function is reset and the compare-equal register is set using Input Compare Position. This value is interpreted as being in CS units if Covert is TRUE and Motor Number is defined in Coord Number. Otherwise this value is interpreted as being motor position in encoder counts. Output Compare Position is a persistent copy of Input Compare Position when Enable Compare was TRUE. The occurance of the compare-equal condition is indicated by Compare-Equal Bool being TRUE. This does not reset the latched condition.

When Encoder Number's captures a position Position Captured is TRUE and the encoder's capture register is queried and converted into Capture Position in motor position encoder counts or Coord Number CS units. If Motor Number is not defined in Coord Number or Convert Bool is FALSE the Capture Position is motor position in encoder counts . If Convert Bool is TRUE and Motor Number is defined in Coord Number the value is in CS units.



Device Number i32 (0) Provided by PmacDevOpen.

Encoder Number i32 (1-16) (1) Encoder to fetch capture position register or reset the compare-equal function for.

Enable Compare Bool (F) When TRUE New Output is TRUE and the encoder's compare-equal register is computed and set based on Input Compare Position. If Convert is FALSE or Motor Number is not defined in Coord Number Input Compare Position is interpreted as motor position in encoder counts. If Convert is TRUE Input Compare Position is interpreted in CS units. In all cases the encoder's compare-equal functions are reset and enabled. Furthermore, because the compare-equal register cannot be read once written,

Output Compare Position maintains a persistent copy of the last written value.

Input Compare Position Dbl Used to set Encoder Number's compare-equal register. This value is interpreted as being in CS units if Covert is TRUE and Motor Number is defined in Coord Number. Otherwise this value is interpreted as being motoro position in encoder counts. Home offsets are removed prior to setting the encoders actual register value. Limitations associated with 24 bit rollowver are not handled by this VI.

Coord Specify Cluster Cluster defining the motor, CS, and conversion state applied when resetting a compare-equal operation of when converting a capture position.

Position Captured Bool (F) This is Encoder Number's capture status flag. When TRUE a capture has been performed and Capture Position contains the capture position in motor position encoder counts or CS units. The reading of the encoder capture register by this VI resets the capture status flag.

Capture Position Dbl In all cases this value is the encoder Capture Position from the last call with Get Capture TRUE. All HOME offsets are removed subject to the roll-over limitations of a 24 bit capture register.
If Motor Number is not defined in Coord Number or Convert Bool is FALSE the value is motor position in encoder counts . If Convert Bool is TRUE and Motor Number is defined in Coord Number the value is in CS units.

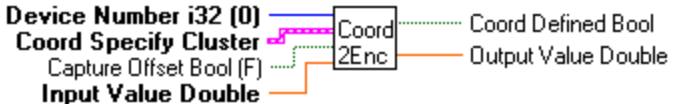
Compare-Equal Bool (F) This is Encoder Number's compare-equal status flag. When TRUE a latched compare-equal condition occured and Output Compare Position contains the value of Input Compare Position set by the VI when Enable Compare was TRUE.

Output Compare Position Dbl This is a persistent copy of Input Compare Position.

PmacEncoderToEncoder

This VI converts Input Value in either CS units or motor position in encoder counts to an absolute encoder position for compare-equal operations.

Coord Specify Cluster specifies a motor within a CS and an attempt to convert Input Value from CS units to encoder position. If the motor is not defined in the CS Input Value is assumed to be motor position in encoder counts and Output Value is encoder position. If the motor is defined and Convert is TRUE Coord Defined is TRUE and Output Value is scaled from CS units to encoder position.



Input Value Double Motor position in encoder counts or possibly CS units convert to compare-equal encoder position.

Device Number i32 (0) Provided by PmacDevOpen

Coord Specify Cluster Cluster defining the motor, CS, and conversion state to be applied

Capture Offset Bool (F) Force a capture of the encoder to motor position offset. This should be done after a home operation.

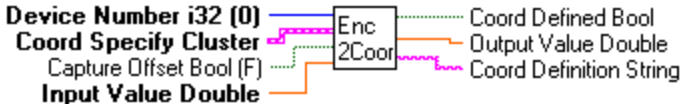
Output Value Double Input value in CS units converted to encoder position if the motor is defined in the CS. Otherwise Input Value is assumed to be motor position in encoder counts and this value is encoder position.

Coord Defined Bool If Motor Number is defined in Coord Number and Convert Bool is TRUE this is TRUE.

PmacEncoderToCoord

This VI converts Input Value (Servo Position or Capture Postion) from absolute encoder position to either CS units or motor position in encoder counts.

Coord Specify Cluster specifies a motor within a CS and an attempt to convert Input Value from encoder position to CS units. If the motor is not defined in the CS Output Value is motor position in encoder counts. If the motor is defined and Convert is TRUE Coord Defined is TRUE and Output Value is in CS units. Coord Definition is a string specifying Output Value units as "Encoder" or the CS definition of the motor.



Input Value Double Position Capture encoder value or servo position to convert to motor position and possibly convert to CS units

Device Number i32 (0) Provided by PmacDevOpen

Coord Specify Cluster Cluster defining the motor, CS, and conversion state to be applied

Capture Offset Bool (F) Force a capture of the encoder to motor position offset. This should be done after a home operation.

Output Value Double Input value in encoder position converted to CS units if the motor is defined in the CS.

Otherwise this value is motor position in encoder counts.

Coord Defined Bool If Motor Number is defined in Coord Number and Convert Bool is TRUE this is TRUE.

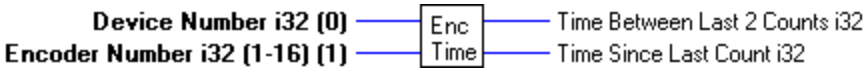
Coord Definition String String defining the motor within the CS. "Encoder" if not defined.

PmacEncoderRegTime

Query PMAC for the two interpolation timers for 1/T sub-count resolution for Encoder Number.

There are two optional methods on PMAC for achieving sub-count resolution with incremental feedback. The first is called 1/T decoding. Each encoder channel has two timer registers associated with it. The first register, Time Between Last 2 Counts, holds the time between the last two encoder transitions. Velocity is estimated as being inversely proportional to this time -- a very accurate estimation, particularly at low speeds.

The second timer, Time Since Las Count, holds the time since the last transition. Fractional distance traveled since the last transition is estimated as the value of the second timer divided by the value of the first timer. This interpolation provides added smoothness to low speed moves, but it does not provide accurate interpolation at rest.



Device Number i32 (0) Provided by PmacDevOpen.

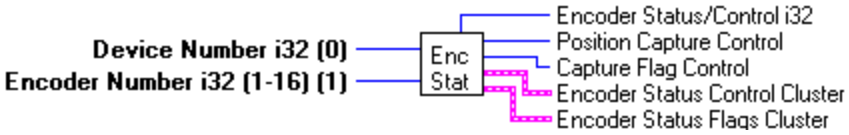
Encoder Number i32 (1-16) (1) Encoder to fetch the current encoder count timer interpolation values for.

Time Since Last Count i32 This value is the encoder's time since the last transition. Fractional distance traveled since the last transition is estimated as this value divided by the value of theTime Between Last 2 Counts. This interpolation provides added smoothness to low speed moves, but it does not provide accurate interpolation at rest.

Time Between Last 2 Counts i32 This is the time between the last two encoder transitions. Velocity is estimated as being inversely proportional to this time -- a very accurate estimation, particularly at low speeds.

PmacEncoderRegStat

Fetch the encoder control/status word for Encoder Number and parse it into its pieces. Encoder Status/Control i32 is the integer representation of the register. Position Capture Control can be used with PmacEncoderCaptureControl. Capture Flag Control can be used with PmacEncoderCaptureFlag. Encoder Status Control Cluster can used with PmacEncoderStatControl. Encoder Status Flag Cluster can be used with PmacEncoderStatFlags.



Device Number i32 (0) Provided by PmacDevOpen.

Encoder Number i32 (1-16) (1) Encoder to fetch the current encoder status for.

Encoder Status/Control i32 i32 representation of encoder status word.

Encoder Status Flags Cluster This cluster contains items for displaying encoder status bits that indicate the status of encoder flags and compare-equal and capture conditions.

Encoder Status Control Cluster This cluster contains items for displaying and modifying encoder control bits that tell PMAC how to handle the compare-equal function and several encoder configuration options.

Capture Flag Control This parameter determines which of the "Flag" inputs will be used for position capture (if one is used -- see I902 etc.):

Setting	Meaning
0	HMFLn (Home Flag n)
1	-LIMn (Negative Limit Signal n)
2	+LIMn (Positive Limit Signal n)
3	FAULTn (Amplifier Fault Signal n)

This parameter is typically set to zero, because in actual use, the +/-LIMn and FAULTn flags create other effects that usually interfere with what is trying to be accomplished by the position capture. If you wish to capture on the +/-LIMn or FAULTn flags, you must either disable their normal functions with Ix25, or use a channel n where none of the flags is used for the normal axis functions.

Position Capture Control This parameter determines which signal or combination of signals (and which polarity) triggers a position capture of the counter for encoder n. If a flag input (home, limit, or fault) is used, I903 (etc.) determines which flag. Proper setup of this variable is essential for a successful home search, which depends on the position-capture function. The following settings may be used:

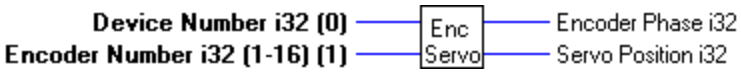
Setting	Meaning
0	Software Control
1	Rising edge of CHCn (third channel)
2	Rising edge of Flag n (as set by Flag Select)
3	Rising edge of [CHCn AND Flag n]
4	Software Control
5	Falling edge of CHCn (third channel)
6	Rising edge of Flag n (as set by Flag Select)
7	Rising edge of [CHCn/ AND Flag n]

8 Software Control
9 Rising edge of CHCn (third channel)
10 Falling edge of Flag n (as set by Flag Select)
11 Rising edge of [CHCn AND Flag n/]
12 Software Control
13 Falling edge of CHCn (third channel)
14 Falling edge of Flag n (as set by Flag Select)
15 Rising edge of [CHCn/ AND Flag n/]

Note that several of these values are redundant. To do a software-controlled position capture, preset this parameter to 0 or 4; when the parameter is then changed to 8 or 12, the capture is triggered (this is not of much practical use).

PmacEncoderRegServo

Query PMAC for the two position registers containing commutation phase and servo position. Servo Position is actual encoder position in counts referenced to a power-up/reset position of zero. Encoder Phase is used internally for commutation.



Device Number i32 (0) Provided by PmacDevOpen.

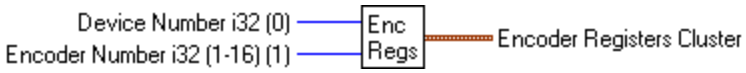
Encoder Number i32 (1-16) (1) Encoder to fetch the current encoder counts for.

Servo Position i32 The encoder Servo position register is 2 * Encoder counts with the LSB the direction bit. This output value is en actual encoder position referenced to a power-up/reset position of zero.

Encoder Phase i32 PMAC provides sophisticated commutation capabilities that require the use of this register.

PmacEncoderRegisters

Query PMAC for all registers for Encoder Number. Assemble the values into a PmacEncoderRegisters Cluster.



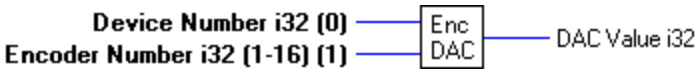
Device Number i32 (0) Provided by PmacDevOpen.

Encoder Number i32 (1-16) (1) Encoder to fetch the various register for.

Encoder Registers Cluster This cluster contains the values of the various encoder registers for use in your application.

PmacEncoderRegDAC

Query PMAC for the current value of the DAC associated with Encoder Number. The raw register uses the upper 16 bits of the 24 bit register. DAC Value shifts the 16 bit value down 8 bits.



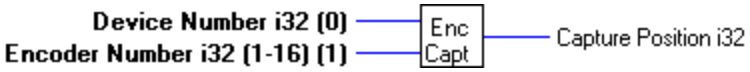
I32 **Encoder Number i32 (1-16) (1)** Encoder to fetch DAC value for.

I32 **Device Number i32 (0)** Provided by PmacDevOpen.

I32 **DAC Value i32** Current value of the DAC associated with the specified encoder. The raw register uses the upper 16 bits of the 24 bit register. This output value shifts the 16 bit value down 8 bits.

PmacEncoderRegCapture

Query PMAC for the Capture Position register. Capture Position is actual encoder position at the time of the capture trigger in encoder counts referenced to a power-up/reset position of zero.



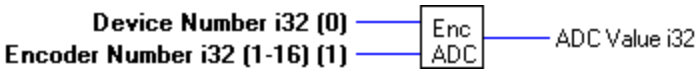
I32 **Device Number i32 (0)** Provided by PmacDevOpen.

I32 **Encoder Number i32 (1-16) (1)** Encoder to fetch capture position register for.

I32 **Capture Position i32** Actual encoder position at the time of the capture trigger in encoder counts referenced to a power-up/reset position of zero.

PmacEncoderRegADC

Query PMAC for the current value of the ADC associated with Encoder Number. The raw register uses the upper 16 bits of the 24 bit register. ADC Value shifts the 16 bit value down 8 bits.



I32 **Device Number i32 (0)** Provided by PmacDevOpen.

I32 **Encoder Number i32 (1-16) (1)** Encoder to fetch ADC value for.

I32 **ADC Value i32** Current value of the ADC associated with specified encoder. The raw register uses the upper 16 bits of

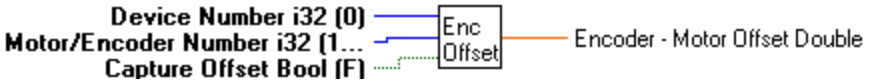
the 24 bit register. This output value shifts the 16 bit value down 8 bits.

PmacEncoderOffset

Query PMAC for the encoder to motor offset captured during a home operation for Encoder/Motor Number. This assumes that encoder 1 is defined for motor 1, etc.

Encoder-Motor Offset provides a reference for using the encoder position-capture and position-compare registers. These registers are referenced to the encoder zero position, which is the power-up position, not the home (motor zero) position. This value is the difference between the two positions plus the home offset Ix26.

This value should be subtracted from encoder position (usually from position capture) to get motor position, or added to motor position to get encoder position (usually for position compare).



Device Number i32 (0) Provided by PmacDevOpen

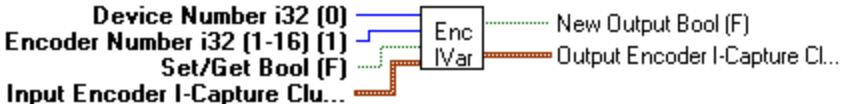
Motor/Encoder Number i32 (1-8) (1) Motor/Encoder number whose encoder to motor position offset is fetched. This assumes that encoder 1 is defined for motor 1, etc.

Capture Offset Bool (F) Force a capture of the encoder to motor position offset. This should be done after a home operation.

Encoder - Motor Offset Double This value provides a reference for using the encoder position-capture and position-compare registers. These registers are referenced to the encoder zero position, which is the power-up position, not the home (motor zero) position. This value is the difference between the two positions plus the home offset Ix26. This value should be subtracted from encoder position (usually from position capture) to get motor position, or added to motor position to get encoder position (usually for position compare).

PmacEncoderIVarCapture

Follow PMACView's standard I-Variable VI architecture. When Set/Get is TRUE the Encoder I-Variables for the specified Encoder Number are set. Otherwise they are fetched from PMAC and provided by Output Encoder I-Capture Cluster with New Output TRUE. Set/Get is not required and defaults to a Get operation.



Device Number i32 (0) Provided by PmacDevOpen.

Encoder Number i32 (1-16) (1) Encoder number whose I-Variables are addressed. This may or may not correspond to a motor.

Set/Get Bool (F) When TRUE PMAC is set using Input Encoder I-Capture Cluster. When FALSE Output Encoder I-Capture Cluster contains the current PMAC definitions.

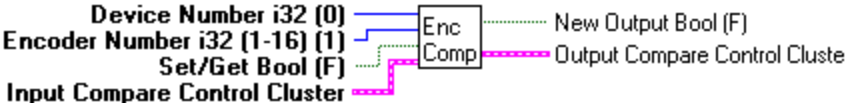
Input Encoder I-Capture Cluster This cluster contains items for displaying and modifying an encoder's position capture trigger I-Variables.

New Output Bool (F) TRUE when Output Encoder I-Capture Cluster contains new data.

Output Encoder I-Capture Cluster This cluster contains items for displaying and modifying an encoder's position capture trigger I-Variables.

PmacEncoderCompareConfig

Follow PMACView's standard I-Variable VI architecture. When Set/Get is TRUE the Input Compare Control bits for the specified Encoder Number are set. Otherwise they are fetched from PMAC and provided by Output Compare Control Cluster with New Output TRUE. Set/Get is not required and defaults to a Get operation.



Device Number i32 (0) Provided by PmacDevOpen.

Input Compare Control Cluster This cluster contains items for displaying and modifying encoder control bits that tell PMAC how to handle the compare-equal function.

Encoder Number i32 (1-16) (1) Encoder number whose compare-equal control bits will be set by Encoder Compare Control Cluster This may or may not correspond to a motor.

Set/Get Bool (F) When TRUE Encoder Number is set using Input Compare Control Cluster. When FALSE Output Compare Control Cluster contains the current PMAC definitions.

Output Compare Control Cluste This cluster contains items for displaying and modifying encoder control bits that tell PMAC how to handle the compare-equal function.

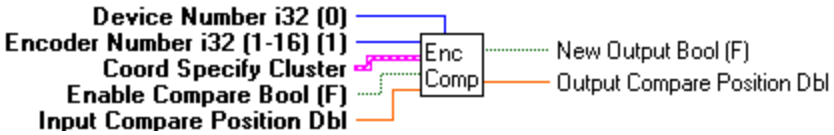
New Output Bool (F) TRUE when Output Compare Control Cluster contains new data.

PmacEncoderCompare

This VI reset Encoder Number's compare-equal function and set the position register using Input Compare Position when Enable Compare is TRUE. This value is interpreted as being in CS units if Covert is TRUE and Motor Number is defined in Coord Number. Otherwise this value is interpreted as being motor position in encoder counts. Home offsets are removed prior to setting the encoders actual register value.

Limitations associated with 24 bit rollowver are not handled by this VI.

Output Compare Position is a persistent copy of Input Compare Position when Enable Compare was TRUE.



Device Number i32 (0) Provided by PmacDevOpen.

Encoder Number i32 (1-16) (1) Encoder to set compare-equal position register for.

Input Compare Position Dbl Used to set Encoder Number's compare-equal register. This value is interpreted as being in CS units if Covert is TRUE and Motor Number is defined in Coord Number. Otherwise this value is interpreted as being motor position in encoder counts. Home offsets are removed prior to setting the encoders actual register value.
Limitations associated with 24 bit roll-over are not handled by this VI.

Enable Compare Bool (F) When TRUE New Output is TRUE and the encoder's compare-equal register is computed and set based on Input Compare Position. If Convert is FALSE or Motor Number is not defined in Coord Number Input Compare Position is interpreted as motor position in encoder counts. If Convert is TRUE Input Compare Position is interpreted in CS units. In all cases the encoder's compare-equal functions are reset and enabled. Furthermore, because the compare-equal register cannot be read once written, Output Compare Position maintains a persistent copy of the last written value.

Coord Specify Cluster Cluster defining the motor, CS, and conversion state to be applied to Input Compare Value when producing a properly referenced compare-equal value for the encoder.

Output Compare Position Dbl This is a persistent copy of Input Compare Position

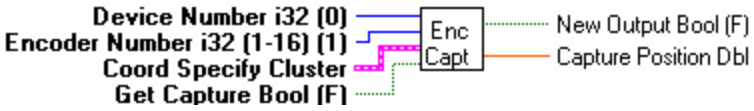
New Output Bool (F) TRUE when Output Compare Value contains new data.

PmacEncoderCapture

This VI queries and converts Encoder Number's Capture Position into motor position encoder counts or Coord Number CS units when Get Capture is TRUE. If Motor

Number is not defined in Coord Number or Convert Bool is FALSE the Capture Position is motor position in encoder counts . If Convert Bool is TRUE and Motor Number is defined in Coord Number the value is in CS units.

When Get Capture is TRUE and a new Capture Position is produced New Output is TRUE. In all cases this value is the encoder Capture Position from the last call with Get Capture TRUE. All HOME offsets are removed subject to the roll-over limitations of a 24 bit capture register.



Device Number i32 (0) Provided by PmacDevOpen.

Encoder Number i32 (1-16) (1) Encoder to fetch capture position register for.

Get Capture Bool (F) When TRUE the New Output is TRUE and encoder capture register is read and converted to Capture Position in motor position encoder units or CS units. Otherwise the last value read is persistently maintained by Capture Position.

Coord Specify Cluster Cluster defining the motor, CS, and conversion state to be applied to encoder register value to produce Capture Position.

Capture Position Dbl In all cases this value is the encoder Capture Position from the last call with Get Capture TRUE. All HOME offsets are removed subject to the roll-over limitations of a 24 bit capture register.
If Motor Number is not defined in Coord Number or Convert Bool is FALSE the value is motor position in encoder counts . If Convert Bool is TRUE and Motor Number is defined in Coord Number the value is in CS units.

New Output Bool (F) TRUE when Capture Position contains new data.

Indicator and Control Clusters

PmacEncoderStatFlags

This cluster contains items for displaying encoder status bits that indicate the status of encoder flags and compare-equal and capture conditions.

Encoder Status Flags Cluster This cluster contains items for displaying encoder status bits that indicate the status of encoder flags and compare-equal and capture conditions.

Compare-Equal M116->X:$C000,16,1 ; Compare-equals flag for encoder 1
This compare-equal signal is always copied into the compare-equal flag (M116 here) that is available for PMAC internal use. If you are using this flag internally, make sure that the signal is latched (M111=1), or you will probably miss it. For interrupting the host (edge-triggered), you will probably want the signal transparent.

Position Captured M103->X:$C003,0,24,S ; Encoder 1 24-bit position capture register
M117->X:$C000,17 ; Encoder 1 position-capture flag
This bit goes TRUE when the trigger condition has gone TRUE; it goes FALSE when the capture register is read (when M103 is used in an expression). As long as the bit is true, the capture function is disabled; you must read the capture register to re-enable the capture function.

Count Error M118->X:$C000,18,1 ; Count error flag for encoder 1
If an illegal encoder transition (both channels changing on the same SCLK cycle) does get through -- or around, if bypassed -- the delay filter, and to the decoder, a count-error flag (M118 here) is set, noting a loss of position information.

C Channel Status Quadrature encoders provide an index channel to indicate revolutions of the encoder.

This flag is TRUE when the channel is TRUE.

Home Flag A home switch may be normally open or normally closed; open is high (1 = TRUE), and closed is low (0 = FALSE). The polarity of the edge that causes the home position capture is programmable with Encoder I-Variables 2 and 3 (I902 and I903 for HMFL1).

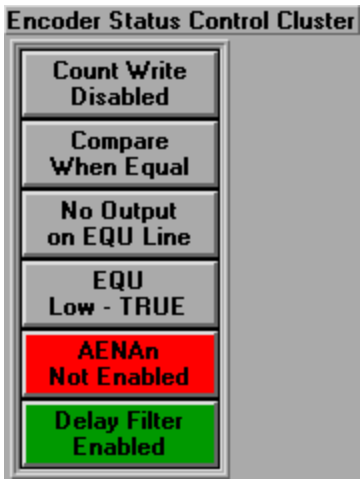
Limit- Flag When assigned for the dedicated uses, these signals provide important safety and accuracy functions. +LIMn and -LIMn are direction-sensitive over-travel limits, that must be actively held low (sourcing current from the pins to ground) to permit motion in their direction.
The direction sense of +LIMn and -LIMn is the opposite of what many peo-ple would consider intuitive. That is, +LIMn should be placed at the nega-tive end of travel, and -LIMn should be placed at the positive end of travel.

Limit+ Flag When assigned for the dedicated uses, these signals provide important safety and accuracy functions. +LIMn and -LIMn are direction-sensitive over-travel limits, that must be actively held low (sourcing current from the pins to ground) to permit motion in their direction.
The direction sense of +LIMn and -LIMn is the opposite of what many peo-ple would consider intuitive. That is, +LIMn should be placed at the nega-tive end of travel, and -LIMn should be placed at the positive end of travel.

Fault Flag This flag takes a signal from the amplifier so PMAC knows when the amplifier is having problems, and can shut down action. The polarity is programmable with I-variable Ix25 (I125 for motor #1) and the return signal is analog ground (AGND). FAULT1 is pin 49. With the default setup, this signal must actively be pulled low for a fault condition. In this setup , if nothing is wired into this input, PMAC will consider the motor not to be in a fault condition.

PmacEncoderStatControl

This cluster contains items for displaying and modifying encoder control bits that tell PMAC how to handle the compare-equal function and several encoder configuration options.



Encoder Status Control Cluster This cluster contains items for displaying and modifying encoder control bits that tell PMAC how to handle the compare-equal function and several encoder configuration options.

Count Write Enable For internal PMAC use.

Compare Latch M111->X:$C000,11,1 ;
Compare flag latch control bit for encoder 1
The flag-latch control bit (M111 here) controls whether the compare-equal signal is transparent -- TRUE only when the positions are actually equal -- or latched -- TRUE until actively reset. The signal is transparent if this control bit is FALSE, and latched if the control bit is TRUE. To clear a latched flag, take the control bit to FALSE, then back to TRUE.

Compare Output M112->X:$C000,12,1 ;
Compare output-enable bit for encoder 1
The output-enable bit (M112 here) determines whether the compare-equal flag will be output on the EQU line (TRUE enables). This must be set if you want to use the signal either to interrupt your host or to trigger an external event directly.

EQU Output Invert M113->X:$C000,13,1
; Compare output invert control bit for encoder 1
The output-invert bit (M113 here) determines whether the EQUn output is high-TRUE or low-TRUE (TRUE inverts -- low-true). For host-interrupt pur-poses, you would want this high-TRUE.

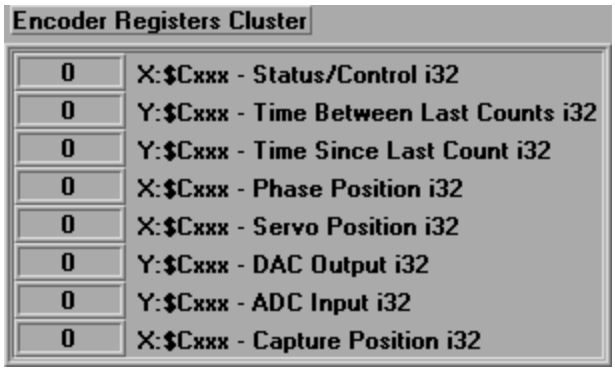
AENAn Most amplifiers have an enable/disable input that permits complete shut-down of the amplifier regardless of the voltage of the command signal. PMAC's AENA line is meant for this purpose. This control bit enables this line.
If you are not using a direc-tion and magnitude

amplifier or voltage-to-frequency converter, you can use this pin to enable and disable your amplifier (wired to the enable line).

Delay Filter Disable Each encoder has a digital delay filter consisting of three cascaded D-flip-flops on each line, with a best two-of-three voting scheme on the outputs of the flip-flops. The flip-flops are clocked by the SCLK signal. This filter does not pass through a state change that only lasts for one SCLK cycle; any change this narrow should be a noise spike. In doing this, the filter delays actual transitions by two SCLK cycles -- a trivial delay in most systems.
This delay filter may be bypassed by setting the Encoder I-Variable 1 (I901, I906, etc.) to 1. This I-Variable maps to this encoder control bit. Bypassing this filter will probably only be done by those users with parallel sub-count interpolation, for which the delay could cause transition errors.

PmacEncoderRegisters

This cluster contains the values of the various encoder registers for use in your application.



Encoder Registers Cluster This cluster contains the values of the various encoder registers for use in your application.

X:$Cxxx - Status/Control i32 i32 representation of encoder status word.

Y:$Cxxx - Time Between Last Counts i32 This is the time between the last two encoder transitions. Velocity is estimated as being inversely proportional to this time -- a very accurate estimation, particularly at low speeds.

Y:$Cxxx - Time Since Last Count i32 This value is the encoder's time since the last transition. Fractional distance traveled since the last transition is estimated as this value divided by the value of theTime

Between Last 2 Counts. This interpolation provides added smoothness to low speed moves, but it does not provide accurate interpolation at rest.

X:$Cxxx - Phase Position i32 PMAC provides sophisticated commutation capabilities that require the use of this register.

X:$Cxxx - Servo Position i32 The encoder Servo position register is 2 * Encoder counts with the LSB the direction bit. This output value is en actual encoder position referenced to a power-up/reset position of zero.

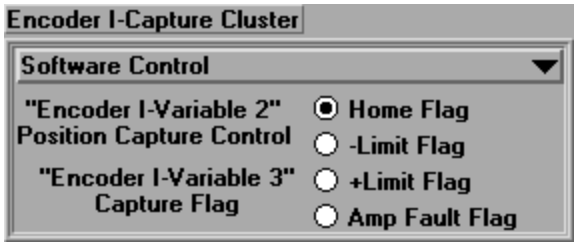
Y:$Cxxx - DAC Output i32 Current value of the DAC associated with the specified encoder. The raw register uses the upper 16 bits of the 24 bit register. This output value shifts the 16 bit value down 8 bits.

Y:$Cxxx - ADC Input i32 Current value of the ADC associated with specified encoder. The raw register uses the upper 16 bits of the 24 bit register. This output value shifts the 16 bit value down 8 bits.

X:$Cxxx - Capture Position i32 Actual encoder position at the time of the capture trigger in encoder counts referenced to a power-up/reset position of zero.

PmacEncoderIVarCapture

This cluster contains items for displaying and modifying Encoder I-Variables 2 and 3. These parameters tell PMAC what trigger conditions and flags to use for position capture.



Encoder I-Capture Cluster This cluster contains items for displaying and modifying Encoder I-Variables 2 and 3. These parameter tell PMAC what trigger conditions and flags to use for position capture.

"Encoder I-Variable 2" Position Capture Control This parameter determines which signal or combination of signals (and which polarity) triggers a position capture of the counter for encoder n. If a flag input (home, limit, or fault) is used, I903 (etc.) determines which flag. Proper setup of this variable

is essential for a successful home search, which depends on the position-capture function. The following settings may be used:

Setting	Meaning
0	Software Control
1	Rising edge of CHCn (third channel)
2	Rising edge of Flag n (as set by Flag Select)
3	Rising edge of [CHCn AND Flag n]
4	Software Control
5	Falling edge of CHCn (third channel)
6	Rising edge of Flag n (as set by Flag Select)
7	Rising edge of [CHCn/ AND Flag n]
8	Software Control
9	Rising edge of CHCn (third channel)
10	Falling edge of Flag n (as set by Flag Select)
11	Rising edge of [CHCn AND Flag n/]
12	Software Control
13	Falling edge of CHCn (third channel)
14	Falling edge of Flag n (as set by Flag Select)
15	Rising edge of [CHCn/ AND Flag n/]

Note that several of these values are redundant. To do a software-controlled position capture, preset this parameter to 0 or 4; when the parameter is then changed to 8 or 12, the capture is triggered (this is not of mu ch practical use).

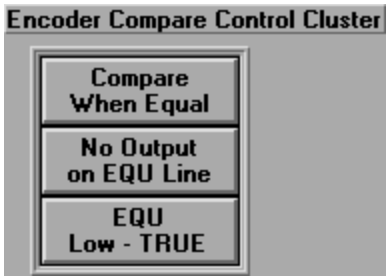
"Encoder I-Variable 3" Capture Flag This parameter determines which of the "Flag" inputs will be used for position capture (if one is used -- see I902 etc.):

Setting	Meaning
0	HMFLn (Home Flag n)
1	-LIMn (Negative Limit Signal n)
2	+LIMn (Positive Limit Signal n)
3	FAULTn (Amplifier Fault Signal n)

This parameter is typically set to zero, because in actual use, the +/-LIMn and FAULTn flags create other effects that usually interfere with what is trying to be accomplished by the position capture. If you wish to capture on the +/-LIMn or FAULTn flags, you must either disable their normal functions with Ix25, or use a channel n where none of the flags is used for the normal axis functions.

PmacEncoderCompareConfig

This cluster contains items for displaying and modifying encoder control bits that tell PMAC how to handle the compare-equal function.



Encoder Compare Control Cluster This cluster contains items for displaying and modifying encoder control bits that tell PMAC how to handle the compare-equal function.

Compare Latch M111->X:$C000,11,1 ;
Compare flag latch control bit for encoder 1
The flag-latch control bit (M111 here) controls whether the compare-equal signal is transparent -- TRUE only when the positions are actually equal -- or latched -- TRUE until actively reset. The signal is transparent if this control bit is FALSE, and latched if the control bit is TRUE. To clear a latched flag, take the control bit to FALSE, then back to TRUE.

Compare Output M112->X:$C000,12,1 ;
Compare output-enable bit for encoder 1
The output-enable bit (M112 here) determines whether the compare-equal flag will be output on the EQU line (TRUE enables). This must be set if you want to use the signal either to interrupt your host or to trigger an external event directly.

EQU Output Invert M113->X:$C000,13,1
; Compare output invert control bit for encoder 1
The output-invert bit (M113 here) determines whether the EQUn output is high-TRUE or low-TRUE (TRUE inverts -- low-true). For host-interrupt pur-poses, you would want this high-TRUE.

PmacEncoderCaptureFlag

This parameter determines which of the "Flag" inputs will be used for position capture (if one is used -- see I902 etc.):

Setting	Meaning
0	HMFLn (Home Flag n)
1	-LIMn (Negative Limit Signal n)
2	+LIMn (Positive Limit Signal n)

3 FAULTn (Amplifier Fault Signal n)

This parameter is typically set to zero, because in actual use, the +/-LIMn and FAULTn flags create other effects that usually interfere with what is trying to be accomplished by the position capture. If you wish to capture on the +/-LIMn or FAULTn flags, you must either disable their normal functions with Ix25, or use a channel n where none of the flags is used for the normal axis functions.

I32 **Capture Flag Control** This parameter determines which of the "Flag" inputs will be used for position capture (if one is used -- see I902 etc.):

Setting Meaning
0 HMFLn (Home Flag n)
1 -LIMn (Negative Limit Signal n)
2 +LIMn (Positive Limit Signal n)
3 FAULTn (Amplifier Fault Signal n)

This parameter is typically set to zero, because in actual use, the +/-LIMn and FAULTn flags create other effects that usually interfere with what is trying to be accomplished by the position capture. If you wish to capture on the +/-LIMn or FAULTn flags, you must either disable their normal functions with Ix25, or use a channel n where none of the flags is used for the normal axis functions.

PmacEncoderCaptureControl

This parameter determines which signal or combination of signals (and which polarity) triggers a position capture of the counter for encoder n. If a flag input (home, limit, or fault) is used, I903 (etc.) determines which flag. Proper setup of this variable is essential for a successful home search, which depends on the position-capture function. The following settings may be used:

Setting Meaning
0 Software Control
1 Rising edge of CHCn (third channel)
2 Rising edge of Flag n (as set by Flag Select)
3 Rising edge of [CHCn AND Flag n]
4 Software Control
5 Falling edge of CHCn (third channel)
6 Rising edge of Flag n (as set by Flag Select)
7 Rising edge of [CHCn/ AND Flag n]
8 Software Control
9 Rising edge of CHCn (third channel)

10	Falling edge of Flag n (as set by Flag Select)
11	Rising edge of [CHCn AND Flag n/]
12	Software Control
13	Falling edge of CHCn (third channel)
14	Falling edge of Flag n (as set by Flag Select)
15	Rising edge of [CHCn/ AND Flag n/]

Note that several of these values are redundant. To do a software-controlled position capture, preset this parameter to 0 or 4; when the parameter is then changed to 8 or 12, the capture is triggered (this is not of much practical use).

U16 **Position Capture Control** This parameter determines which signal or combination of signals (and which polarity) triggers a position capture of the counter for encoder n. If a flag input (home, limit, or fault) is used, I903 (etc.) determines which flag. Proper setup of this variable is essential for a successful home search, which depends on the position-capture function. The following settings may be used:

Setting	Meaning
0	Software Control
1	Rising edge of CHCn (third channel)
2	Rising edge of Flag n (as set by Flag Select)
3	Rising edge of [CHCn AND Flag n]
4	Software Control
5	Falling edge of CHCn (third channel)
6	Rising edge of Flag n (as set by Flag Select)
7	Rising edge of [CHCn/ AND Flag n]
8	Software Control
9	Rising edge of CHCn (third channel)
10	Falling edge of Flag n (as set by Flag Select)
11	Rising edge of [CHCn AND Flag n/]
12	Software Control
13	Falling edge of CHCn (third channel)
14	Falling edge of Flag n (as set by Flag Select)
15	Rising edge of [CHCn/ AND Flag n/]

Note that several of these values are redundant. To do a software-controlled position capture, preset this parameter to 0 or 4; when the parameter is then changed to 8 or 12, the capture is triggered (this is not of much practical use).

PmacFile

\PmacFile - Tools for maintaining ASCII program files, LabVIEW datalog files, and downloading files to PMAC.

VIs

PmacFileDatalog

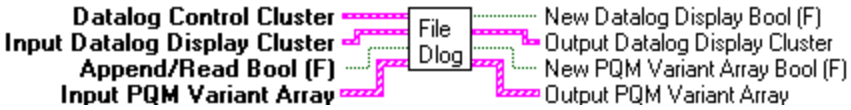
Manage datalog operations for type-neutral PmacPQMVariant Arrays.

Operations as specified by the radio buttons in Datalog Control Cluster are performed when Append/Read is TRUE. A file must be selected prior to executing the operation using the Create/Open button or New File button in the cluster. The file is opened and closed on every transaction. After an operation New Datalog Display is TRUE and Output Datalog Display Cluster contains updated operation status for your application's cluster.

Append operations write Input PQM Variant Array to the end of the file specified in Input Datalog Display Cluster and update Current Record and Num Records in the output cluster. The contents of the Note window are appended with the record.

Read operations read the record specified by Current Record in Input Datalog Display Cluster from the specified file and generate a new Output PQM Variant Array. The availability of new data is indicated by New PQM Variant Array TRUE. Output Datalog Display Cluster increments Current Record and displays the Note, if any, attached to the record. Read operations cannot read past the end of the file and simply read the last record in the file.

To change the data logged by this VI simply change Input and Output PQM Variant Array to your own data type. similar modifications to PmacFileDatalogAppend, Create, and Read are also required.



Input PQM Variant Array Array of PmacPQMVariants to write/append to the specified datalog file.

Datalog Control Cluster PmacFileDatalogControl specifies what operations this VI will perform during its next execution.

Input Datalog Display Cluster Input Datalog Display Cluster contains information about the record to be written or read during the next read or write to a datalog file.

Append/Read Bool (F) Execute the specified Read, Append, or Ignore operation when TRUE.

Output PQM Variant Array Array of PmacPQMVariants read during the last datalog read.

New PQM Variant Array Bool (F) Output PQM Variant Cluster contains new PQM data.

Output Datalog Display Cluster Output Datalog Display Cluster contains information about the record to be written or read during the next read or write to a datalog file.

New Datalog Display Bool (F) Output Datalog Display Cluster contains new information about a datalog file path,

record just read or written, and notes that may have been read.

PmacFileDatalogAppend

When Append Record is TRUE append Input PQM Variant Array to the file specified in Input Datalog Display at the end of the file. Update the Current Record and Num Records in Output Datalog Display Cluster. Indicate the new data by setting new Datalog Display TRUE.

Input PQM Variant Array Array of PmacPQMVariants to append to the datalog file.

Input Datalog Display Cluster Input Datalog Display Cluster contains information about the record to be written during the next append to a datalog file.

Append Record Bool Execute an append operation when TRUE.

Output Datalog Display Cluster Output Datalog Display Cluster contains information about the next record to be written during the next write to a datalog file.

New Datalog Display Bool (F) TRUE when a record was written to the datalog file and Output Datalog Display Cluster indicates the next record to be written

PmacFileDatalogCreate

Create or Open an existing datalog file to store data of type Input PQM Variant Array along with notes. When Create/Open is TRUE use the path in Input Datalog Cluster. If this path is empty or Not A Path display a file selection dialog. When a file name is entered or an existing file is selected the number of records in the file is determined. All updated information is available in Output Datalog Display Cluster and indicated by New Datalog Display TRUE.

Input PQM Variant Array Input PQM Variant Array defines the type of user data that will be stored in the datalog file. The actual record is a cluster containing this array, notes, and other information you might add such as data, time, user name, etc.

Create/Open Bool When TRUE Create a new datalog file to store PmacPQMVariant arrays or Open an existing datalog file containing PmacPQMVariant arrays.

Input Datalog Display Cluster Input Datalog Display

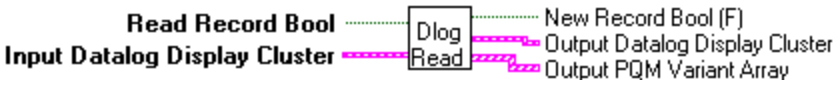
Cluster contains the file path of the datalog file to be created or opened. If the file path is empty or Not A Path a dialog to locate or name a file is displayed.

Output Datalog Display Cluster Output Datalog Display Cluster contains the path of the file to be read or written during the next datalog operation.

New Datalog Display Bool (F) Output Datalog Display Cluster contains new information about the datalog file path and the number of records contained in the file.

PmacFileDatalogRead

When Read Record is TRUE read Output PQM Variant Array from the file specified in Input Datalog Display using Current Record. Update the increment Current Record in Output Datalog Display Cluster and display the Note, if any, stored with the record. Indicate the new data by setting new Datalog Display TRUE.



Input Datalog Display Cluster Input Datalog Display Cluster contains information about the record to be read during the next read from a datalog file.

Read Record Bool Execute the a read record operation when TRUE.

Output PQM Variant Array When New Record is TRUE this array contains type-neutral PmacPQMVariants read from the datalog file.

Output Datalog Display Cluster Output Datalog Display Cluster contains information about the record just read from the datalog file.

New Record Bool (F) TRUE when a record was read from the datalog file and Output Datalog Display Cluster displays any notes associated with the record and Current Record indicates the next record to be read

PmacFileDownLoad

Compile the ASCII motion or PLC program specified by Down Load File Path and down load the compiled code to PMAC. If there is a compile error Down Load Error is TRUE. Down Load Log File Path specifies the path to the compile log file.

Example

If Down Load File Path is PmacTest1.pmc this path is PmacTest1.log. If there is a compile error a pop-up panel displaying the log file appears to detail compile and down load errors.

Device Number i32 (0) Provided by PmacDevOpen

Down Load File Path Path of existing ASCII motion or PLC program to down load to PMAC.
There is no ability to browse for a down file to down load.

Down Load Log File Path Path to the down load compile log. If the Down Load File Path is PmacTest1.pmc this path is PmacTest1.log

Down Load Error Bool (F) TRUE if the file doesn't exist or there was an error compiling the program.

PmacFileDownLoadLog

Load the down load compile log file generated by PmacFileDownLoad and display the contents. This panel automatically appears if an error occurs during the down load process. The modal panel must be closed before continuing.

Log File Path Name of Down Load compile log file to display.

PmacFileLoad

Input Path is the file path of an ASCII file to read into Buffer String. If Input Path is empty (default value) or is Not A Path, the VI displays a File dialog box from which you can select a file. Output Path is the file path that was loaded and contained in Buffer String. Output Path is NotAPath if the user canceled the operation.

Error 43 occurs if the user cancels the dialog.

Input Path (dialog if empty) Input Path is the path name of an ASCII file to load. If file path is empty (default value) or is Not A Path, the VI displays a File dialog box from which you can select a file.

Output Path (NotAPath if canceled) Output Path is the path of the file from which Buffer String is loaded. You can use this output to determine the path of a file that you opened using dialog. Output Path returns Not A Path if the user selects Cancel from the dialog box.

Buffer String ASCII data read from the file.

PmacFileSave

Input Path is the file path of an ASCII file to write Buffer String to. If Input Path is empty (default value) or is Not A Path, the VI displays a File dialog box from which you can select a file. Output Path is the file path of the file where Buffer String was saved. Output Path is NotAPath if the user canceled the operation.

Error 43 occurs if the user cancels the dialog.

Input Path (dialog if empty) The path name of the file to save Buffer String to. If Input Path is empty (default value) or is Not A Path, the VI displays a File dialog box from which you can select a file. Error 43 occurs if the user cancels the dialog.

Buffer String Buffer String is the data the VI writes to the file.

Output Path (Not A Path if canceled) Output Path is the path of the file to which the VI wrote Buffer String. You can use this output to determine the path of a file that you open using dialog. new file path returns Not A Path if the user selects Cancel from the dialog box.

PmacFileSpreadsheet

Write Data Array to a spreadsheet file specified by Input File Path. If this path is empty or Not A Path a selection dialog is displayed. Output File Path is the path to the file actually written. The spreadsheet file will contain a column for each element in Data Array. The first row in each column is the tab-delimited text description contained in Column Header. Each successive row has an assumed sample interval of 1.

Input File Path Path to spreadsheet file to write Data Array to. If this is empty or Not A Path a dialog is displayed.

Data Array Double Data Array to be written to spreadsheet file.

Column Header String Optional tab-delimited column header strings

Output File Path Actual file path where Data Array was written.

Indicator and Control Clusters

PmacFileDatalogControl

This cluster defines how PmacFileDatalog will handle requests for service. Read will read a record, Append writes a record, and Ignore take no action. Create/Open allows the user to select an existing file for reading or to append new records to. New file selects a new file for Read or Append operations.

Datalog Control Cluster This cluster defines how PmacFileDatalog will handle requests for service.

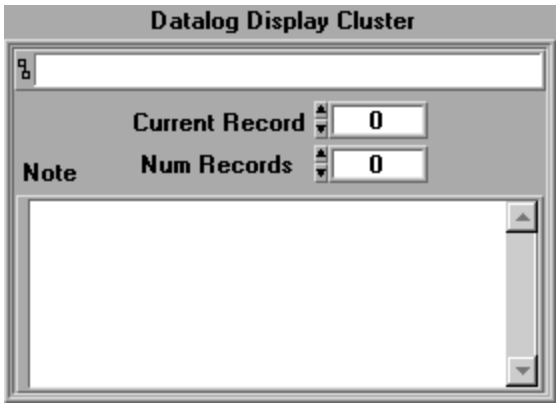
Create/Open Create or Open a datalog file for reading or appending. If the File path in the corresponding Datalog Display Cluster is empty or Not A Path a File Dialog is displayed to select a file. If a new file name is entered a file is created. If an existing file is selected it is opened.

New File Create or Open a new datalog file for reading or appending. A File Dialog is displayed to select a file. If a new file name is entered a file is created. If an existing file is selected it is opened.

Append/Read Operation Select a datalog file operation to be executed during the Read/Append operation. Read will record specified by Current Record on the next operation. Append will write the record to the end of the specified file. Ignore will neither read nor append a record.

PmacFileDatalogDisplay

This cluster works with PmacFileDatalog to specify which datalog file to work with. When a record is appended to a datalog file Current Record reflects the record that will be written. The contents of Note will be written with the record.



Datalog Display Cluster This cluster contains information about the record to be written or read during the next read or write to a datalog file by PmacFileDatalog.

Datalog Path (dialog if empty) Full path name for the datalog file. Only required when creating a new file (Mode = Create).

Current Record Number of datalog record to be read or record just written.

Num Records Number of records in datalog file.

Note Text note to be written with record to datalog file

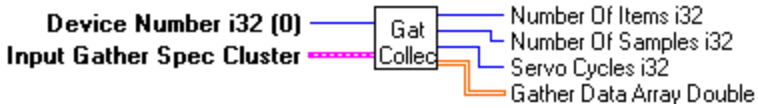
PmacGather

\PmacGather - This collection of VI's allow the user to select motion variables to control the collection and conversion of data into standard LabVIEW analysis formats. Collected data can be output to files in tab delimited format for export to programs such as Matlab or Excel.

VIs

PmacGatherCollect

Collect the gather buffer and scale the data using each items scale factor. The data in Gather Data Array is a two dimensional array of doubles with Number of Items columns and Number of Samples rows. In this format the data can be written to a spreadsheet or processed by many different LabVIEW data analysis VIs.



Device Number i32 (0) Provided by PmacDevOpen

Input Gather Spec Cluster Cluster containing servo sample rate and array of clusters of Address Items describing item, address, and scale factor.

Number Of Samples i32 Number of samples gathered

Number Of Items i32 Number of source items gathered.

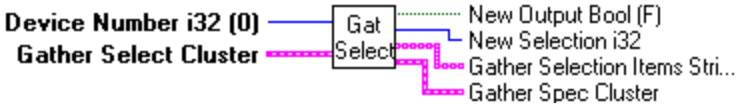
Gather Data Array Double Two dimensional array of gathered data. There are Number of Sources columns and Number of Samples rows. The data for each source item is scaled using the scale factor for that item.

Servo Cycles i32 Sample rate in Servo Cycles.

PmacGatherSelect

Maintains a PmacGatherSelect Cluster and builds a PmacGatherSpec cluster to define gather operations. You can builds a list in four ways. Select an item and Motor/CS number, P-Variable, Q-Variable, of define a Custom Gather Specification. Click the associated -> button to add the item to the list on the right. The gather sample rate is defined as a number of Servo Cycles. All items are gathered at the same sample rate. Items selected in the list can be deleted using the Remove button.

New Output is TRUE when an item is added to the list with a -> button or removed from the list with Remove. New selection identifies the selected item in the gather list. Gather Selection Items String Array define the contents of the gather list. Gather Spec Cluster is an internal data type used by other PmacGather VIs to setup PMAC and collect the gathered data.



Gather Select Cluster Build a list of Address Items to be gathered.

Device Number i32 (0) Provided by PmacDevOpen

New Output Bool (F) TRUE when any -> or Remove button in Gather Select Cluster is clicked. This indicates that New Selection, Gather Selection Items, and Gather Spec Cluster all contain new data.

Gather Spec Cluster Cluster containing servo sample rate and array of clusters of Address Items describing item, address, and scale factor.

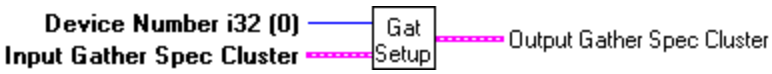
Gather Selection Items String Array Array of strings for items in Gather Selection Items ring.

New Selection i32 Index of currently selected it in Gather Selection Items ring.

PmacGatherSetup

Use the information in Input Gather Spec Cluster to setup a gather operation on PMAC. Output Gather Spec Cluster should be wired to PmacGatherStart, PmacGatherStop, and PmacGatherCollect to sequence operations and so that they can get the information they require for their operation.

The actual setup can also be done using Pewin32, PmacTerminal, or your motion/PLC programs. This is not recommended if you intend to use PmacGatherCollect to retrieve the gathered data. These methods require intimate knowledge of PMAC's internal architecture and are automatically handled by this VI.



Device Number i32 (0) Provided by PmacDevOpen

Input Gather Spec Cluster Cluster containing servo sample rate and array of clusters of Address Items describing item, address, and scale factor.

Output Gather Spec Cluster Cluster containing servo sample rate and array of clusters of Address Items describing item, address, and scale factor.

PmacGatherSpreadsheet

Write Gather Data Array to spreadsheet file specified by Input File Path. If this path is empty or Not A Path a selection dialog is displayed. Output File Path is the path to the file actually written. The spreadsheet file will contain a column for each item gathered. The first row in each column is a text description of the gathered item. Each successive row has an assumed sample interval of Servo Cycles.

Gather Spec Cluster Cluster containing servo sample rate and array of clusters of Address Items describing item, address, and scale factor.

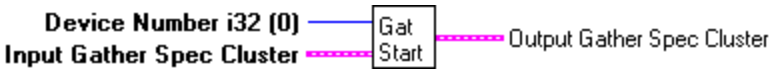
Input File Path Path to spreadsheet file to write Gather Data Array to. If this is empty or Not A Path a dialog is displayed.

Gather Data Array Double Gather Data Array to be written to spreadsheet file.

Output File Path Actual file path where Gather Data Array was written.

PmacGatherStart

Start a previously defined gather operation. You should immediately start the desired motion after this VI executes. You can eliminate this VI if you start gathering by using the PMAC program command "define gather" in your program.



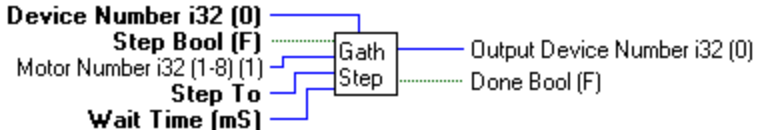
Device Number i32 (0) Provided by PmacDevOpen

Input Gather Spec Cluster Cluster containing servo sample rate and array of clusters of Address Items describing item, address, and scale factor.

Output Gather Spec Cluster Cluster containing servo sample rate and array of clusters of Address Items describing item, address, and scale factor.

PmacGatherStep

This VI forces Motor Number to immediately have a desired encoder position of Step To. This is done by scaling Step To by Ix08 and writing the value to an internal PMAC register and results in a closed loop step response. The VI waits for Wait Time before continuing by setting Done TRUE.



Device Number i32 (0) Provided by PmacDevOpen

Step To Motor position to step to in encoder counts.

Wait Time (mS) Time in mS to wait after executing step.

Motor Number i32 (1-8) (1) Motor Number to Step

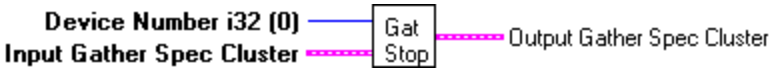
Step Bool (F) Move Motor Number to the position specified by Step To when this is TRUE. Wait for Wait Time before continuing.

Done Bool (F) TRUE when a step was executed and completes Wait Time.

Output Device Number i32 (0) Copy of Device Number for sequencing.

PmacGatherStop

Stop an executing gather operation. You can eliminate this VI if you stop gathering by using the PMAC program command "end gather" in your program.



Device Number i32 (0) Provided by PmacDevOpen

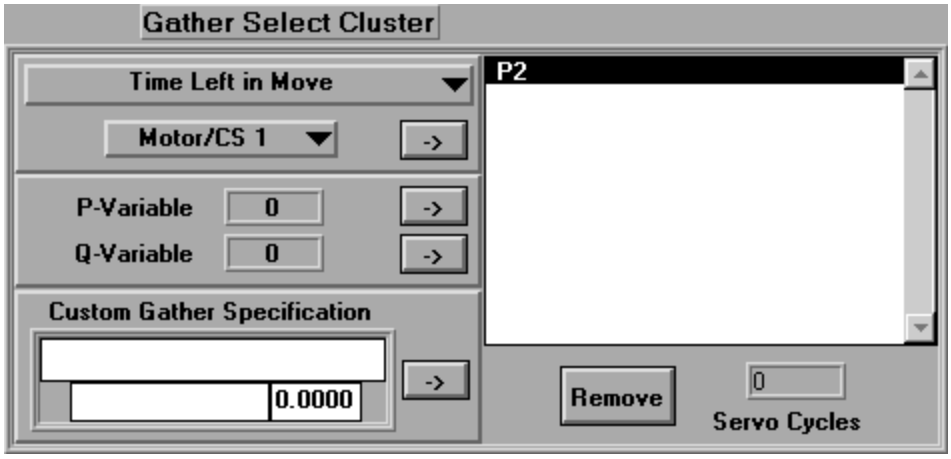
Input Gather Spec Cluster Cluster containing servo sample rate and array of clusters of Address Items describing item, address, and scale factor.

Output Gather Spec Cluster Cluster containing servo sample rate and array of clusters of Address Items describing item, address, and scale factor.

Indicator and Control Clusters

PmacGatherSelect

This cluster allows the user to build a list of Address Items. You can builds a list in four ways. Select an item and Motor/CS number, P-Variable, Q-Variable, of define a Custom Gather Specification. Click the associated -> button to add the item to the list on the right. The gather sample rate is defined as a number of Servo Cycles. All items are gathered at the same sample rate. Items selected in the list can be deleted using the Remove button.



Gather Select Cluster Build a list of Address Items to be gathered.

Motor/CS Number Select a Motor/CS for Address Item

Motor/CS Item Description Allow selection of motor/CS states defined in PmacAddressMotors

Gather Selection Items Description list of items to be gathered. Can select up to 24 items.

Add Motor/CS Item Add a Motor/CS item

Remove Remove the item selected in the list box above

Servo Cycles Define how many servo cycles between gather samples.

Custom Gather Specification Specify a description, address, and scale factor for a custom Address Item

- **Address Item Description** Text description of **Address Item**
- **Address Item Address** Address of **Address Item**
- **Address Item Scale** Scale factor for **Address Item**

Add Custom Add a custom Address Item

Add P-Var Add a P-Variable

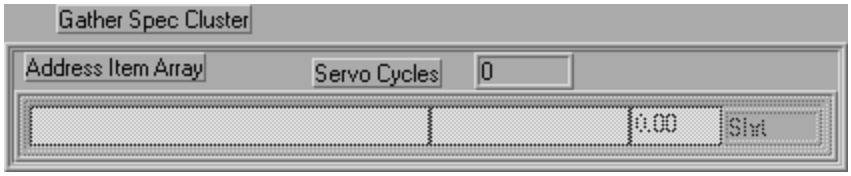
P-Variable P-Variable to gather

Add Q-Var Add a Q-Variable

Q-Variable Q-Variable to gather

PmacGatherSpec

Cluster containing servo sample rate and array of clusters of Address Items describing item, address, and scale factor.



Gather Spec Cluster Cluster containing servo sample rate and array of clusters of Address Items describing item, address, and scale factor.

Servo Cycles Define how many servo cycles between gather samples.

Address Item Array Array of Address Item clusters

Address Item Cluster Specify a description, address, and scale factor for a Address Item

- **Address Item Description** Text description of Address Item
- **Address Item Address** Address of Address Item

Address Item Scale Scale factor for Address Item

PmacGlobal

\PmacGlobal - General PMAC setup and configuration are provided by this collection for VIs. These capabilities are used for the development of supervisory VIs.

VIs

PmacGlobalBufferClose

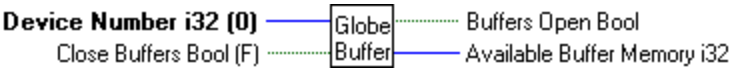
Delete any open gather buffer and close any open program, PLC, or rotary buffers. The output Device Number can be used to sequence operations.

Device Number i32 (0) Provided by PmacDevOpen

Device Number i32 (0) Passed from input Device Number after buffers are closed

PmacGlobalBufferSize

Monitor and control PMAC buffer space during system development. Buffers Open is TRUE if an open prog, open rotary, or open PLC command has been executed and the corresponding buffer has not been closed yet. Available Buffer Memory specifies how much buffer space PMAC has left for gathering and programs. A define gather command reserves all available buffer space. If Close Buffers is TRUE the gather buffer is deleted and a close command is sent to PMAC.



Device Number i32 (0) Provided by PmacDevOpen

Close Buffers Bool (F) Send commands to PMAC to close any open buffers.

Available Buffer Memory i32 Amount of available PMAC buffer space.

Buffers Open Bool TRUE if there are any open program, PLC, or rotary buffers open.

PmacGlobalControl

Generate PMAC on-line commands for controlling PMAC program execution and save state. Command Executed Bool is TRUE when any button is clicked in Global Control Cluster.

Device Number i32 (0) Provided by PmacDevOpen

Global Control Cluster PmacGlobalControl Cluster of latched buttons to control PMAC program execution and save state

Command Executed Bool (F) TRUE when any Global Control Cluster button is clicked

PmacGlobalIVarComm

Follow PMACPanel's standard I-Variable VI architecture. When Set/Get is TRUE the global communication I-Variables are set. Otherwise they are fetched from PMAC and provided by Output Global I-Comm Cluster with New Output TRUE. Set/Get is not required and defaults to a Get operation.

Device Number i32 (0) Provided by PmacDevOpen

Input Global I-Comm Cluster Input PmacGlobalIVarComm cluster for communication I-Variables

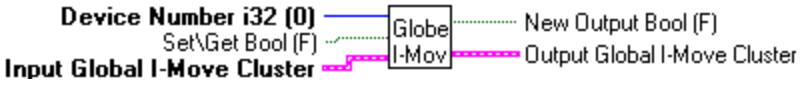
Set\Get Bool (F) When TRUE PMAC is set using Input Global I-Comm Cluster. When FALSE Output Motor I-Comm Cluster contains the current PMAC definitions.

New Output Bool (F) TRUE when Output Global I-Comm Cluster contains new data.

Output Global I-Comm Cluster Output PmacGlobalIVa rComm cluster for communication I-Variables

PmacGlobalIVarMove

Follow PMACPanel's standard I-Variable VI architecture. When Set/Get is TRUE the global movement I-Variables are set. Otherwise they are fetched from PMAC and provided by Output Global I-Move Cluster with New Output TRUE. Set/Get is not required and defaults to a Get operation.



Device Number i32 (0) Provided by PmacDevOpen

Input Global I-Move Cluster Input PmacGlobalIVarMove cluster for move I-Variables

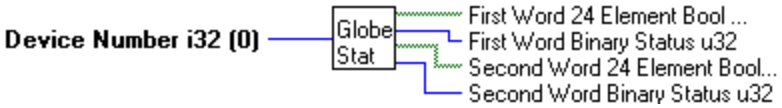
Set\Get Bool (F) When TRUE PMAC is set using Input Global I-Move Cluster. When FALSE Output Motor I-Move Cluster contains the current PMAC definitions.

New Output Bool (F) TRUE when Output Global I-Move Cluster contains new data.

Output Global I-Move Cluster Output PmacGlobalIVarMove cluster for move I-Variables

PmacGlobalStat

Query PMAC for PMAC's global status. Report the two status words as arrays of Booleans and unsigned 32 bit integers.



Device Number i32 (0) Provided by PmacDevOpen

First Word 24 Element Bool Array 24 element Boolean array containing first status word

Second Word 24 Element Bool Array 24 element Boolean array containing second status word

First Word Binary Status u32 integer representation of first word status

Second Word Binary Status u32 integer representation of second word status

PmacGlobalStatBuffer

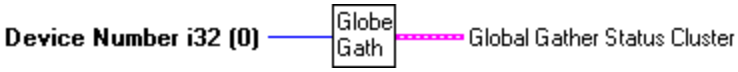
Create a status indicator cluster for the PmacGlobalStatBuffer indicator containing PMAC's global buffer status.

Device Number i32 (0) Provided by PmacDevOpen

Global Buffer Status Cluster Cluster for PmacGlobalStatBuffer

PmacGlobalStatGather

Create a status indicator cluster for the PmacGlobalStatGather indicator containing PMAC's global gather status.

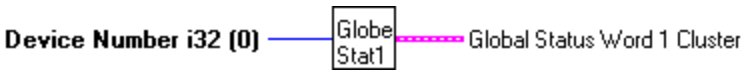


Device Number i32 (0) Provided by PmacDevOpen

Global Gather Status Cluster Cluster for PmacGlobalStatGather

PmacGlobalStatWord1

Create a status indicator cluster for the PmacGlobalStat1 indicator containing PMAC's global status.

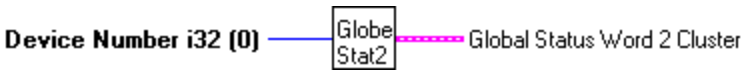


Device Number i32 (0) Provided by PmacDevOpen

Global Status Word 1 Cluster Cluster for PmacGlobalStat1

PmacGlobalStatWord2

Create a status indicator cluster for the PmacGlobalStat2 indicator containing PMAC's global status.



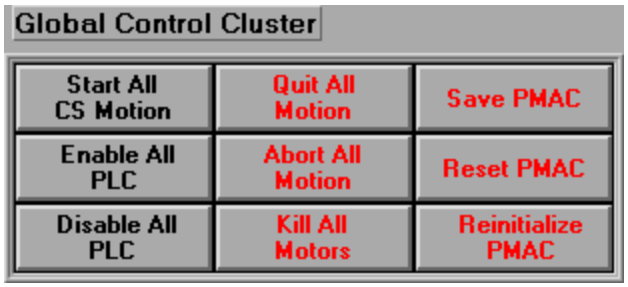
Device Number i32 (0) Provided by PmacDevOpen

Global Status Word 2 Cluster Cluster for PmacGlobalStat2

Indicator and Control Clusters

PmacGlobalControl

Cluster of latched buttons to control PMAC program execution and save state



Global Control Cluster Cluster of latched buttons to control PMAC program execution and save state

Start Motion Programs The command sent to PMAC is ^R.
This command is the equivalent of issuing the R (run) command to all coordinate systems in PMAC. Each active coordinate system (i.e. one that has at least one motor assigned to it) that is to run a program must already be pointing to a motion program (initially this is done with a B{prog num} command).
For multiple cards on a single serial daisy-chain, this command affects all cards on the chain, regardless of the current software addressing.

Enable PLC Programs The command sent to PMAC is ENABLE PLC 0..31.
This command causes PMAC to enable (start executing) the specified PLC program or programs. PLC programs are specified by number, and may be used singularly in this command, in a list (separated by commas), or in a range of consecutively numbered programs.
If a motion or PLC program buffer is open when this command is sent to PMAC, the command will be entered into that buffer for later execution.
I-variable I5 must be in the proper state to allow the PLC program(s) specified in this command to execute.

Disable PLC Programs The command sent to PMAC is ^D.
This command causes all PLC programs to be disabled (i.e. stop executing). This is the equivalent of DISABLE PLC 0..31. It is especially useful if a CMD or SEND statement in a PLC has run amok.
For multiple cards on a single serial daisy-chain, this command affects all cards on the chain, regardless of the current software addressing.

Kill Motion Programs The command sent to PMAC is ^Q.
This command causes any and all motion programs running in any coordinate system to stop executing after the moves that have already been calculated are

finished. Program execution may be resumed from this point with the R (run) or S (step) commands.
For multiple cards on a single serial daisy-chain, this command affects all cards on the chain, regardless of the current software addressing.

Abort The command sent to PMAC is ^A.
This command aborts all motion programs and stops all non-program moves on the card. It also brings any disabled or open-loop motors to an enabled zero-velocity closed-loop state. Each motor will decelerate at a rate defined by its own motor I-variable Ix15. However, a multi-axis system may not stay on its programmed path during this deceleration.
A <CTRL-A> stop to a program is not meant to be recovered from gracefully, because the axes will in general not stop at a programmed point. The next programmed move will not behave properly unless a PMATCH command is given or I14 is set to 1 (these cause PMAC to use the aborted position as the move start position). Alternately, an on-line J= command may be issued to each motor to cause it to move to the end point that was programmed when the abort occurred. Then the program(s) can be resumed with an R (run) command.
To stop a motion sequence in a manner that can be recovered from easily, use instead the Quit (Q or <CTRL-Q>) or the Hold (H or <CTRL-O>) command.
When PMAC is set up to power on with all motors killed (Ix80 = 0), this command can be used to enable all of the motors (provided that they are not commutated by PMAC -- in that case, each motor should be enabled with the $ command).
For multiple cards on a single serial daisy-chain, this command affects all cards on the chain, regardless of the current software addressing.

Kill Motors The command sent to PMAC is ^K.
This command kills all motor outputs by opening the servo loop, commanding zero output, and taking the amplifier enable signal (AENAn) false (polarity is determined by jumper E17) for all motors on the card. If any motion programs are running, they will automatically be aborted. (For the motor-specific K (kill) command, if the motor is in a coordinate system that is executing a motion program, the program execution must be stopped with either an A (abort) or Q (quit) command before PMAC will accept the K command.)
For multiple cards on a single serial daisy-chain, this command affects all cards on the chain, regardless of the current software addressing.

Save Pmac The command sent to PMAC is SAVE.
This command causes PMAC to save its basic setup

parameters into the non-volatile EAROM memory. These setup parameters include all I-variables, the encoder conversion table, and memory locations containing VME and/or dual-ported RAM addressing information. At the next power-on/reset cycle, this information will be copied back into RAM (provided jumper E51 is in its default state -- OFF for PMAC-PC, -Lite, and -VME; ON for PMAC-STD).
The SAVE command is the only time PMAC will write to the EAROM. Jumper E50 must be on for the SAVE command to operate.
PMAC does not provide the acknowledging handshake character to the SAVE command until it has finished the saving operation, a significant fraction of a second later. The host program should be prepared to wait much longer for this character than is necessary on most commands. For this reason, it is usually not a good idea to include the SAVE command as part of a "dump" download of a large file.
If this command is given from the PMAC Executive Program terminal mode while continuously polling windows (such as a position window) are open, the Executive Program may be temporarily confused by the delay and place information in the wrong window.

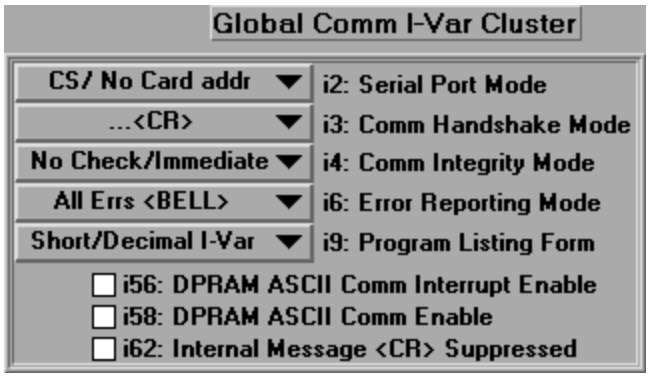
Reset Pmac The command sent to PMAC is $$$.
This command causes PMAC to do a full card reset. The effect of $$$ is equivalent to that of cycling power on PMAC, or taking the INIT/ line low, then high.
With jumper E51 in its default state (OFF for PMAC-PC, -Lite, -VME, ON for PMAC-STD), this command does a standard reset of the PMAC. I-variable values, conversion-table settings, and DPRAM and VMEbus addresses stored in permanent memory (EAROM) by the last SAVE command are reloaded into active memory (RAM).
With jumper E51 in non-default state (ON for PMAC-PC, -Lite, -VME, OFF for PMAC-STD), this command does a reset and re-initialization of the PMAC. Factory default I-variable values, conversion-table settings, and DPRAM and VMEbus addresses stored in the firmware (EPROM) are copied into active memory (RAM). (Values stored in EAROM are not lost; they are simply not used.)
Because this command immediately cause PMAC to enter its power-up/rest cycle, there is no acknowledging character (<ACK> or <LF>) returned to the host.

Reinitialize Pmac The command sent to PMAC is $$$***.
This command performs a full reset of the card and reinitializes the memory. All programs and other

buffers are erased. All I-variables, encoder conversion table entries, and VME and DPRAM addressing parameters are returned to their factory defaults. (Previously SAVEd values for these parameters are still held in EAROM, and can be brought into active memory with a subsequent $$$ command).
M-variable definitions, P-variable values, Q-variable values, and axis definitions are not affected by this command. They can be cleared by separate commands (e.g. M0..1023->*, P0..1023=0, Q0..1023=0, UNDEFINE ALL).
This command is particularly useful if the program buffers have become corrupted. It clears the buffers and buffer pointers so the files can be re-sent to PMAC. Regular backup of parameters and programs to the disk of a host computer is strongly encouraged so this type of recovery is possible. The PMAC Executive program has "Save Full PMAC Configuration" and "Restore Full PMAC Configuration" functions to make this process easy.

PmacGlobalIVarComm

This cluster contains items for displaying and modifying PMAC's communication I-Variables.



Global Comm I-Var Cluster This cluster contains items for displaying and modifying PMAC's communication I-Variables.

i2: Serial Port Mode

i3: Comm Handshake Mode This parameter controls what characters, if any, are used by PMAC to delimit a transmitted line, and whether PMAC issues an acknowledgment (handshake) of a command.
Note: With checksum enabled (I4=1), checksum bytes are added after the handshake character bytes.
Valid values of I3 and the modes they represent are:
0: PMAC does not acknowledge receipt of a valid

command. It returns a <BELL> character on receipt of an invalid command. Messages are sent without beginning or terminating <LF> (line feed); simply as DATA <CR> (carriage return).
1: PMAC acknowledges receipt of a valid <CR>-terminated command with a <LF>; of an invalid command with a <BELL> character. Messages are sent as <LF> DATA <CR> [<LF> DATA <CR> ...] <LF>. (The final <LF> is the acknowledgment of the host command; it does not get sent with a message initiated from a PMAC program [SEND or CMD]). This setting is good for communicating with terminal display programs, such as the PMAC Executive program.
2: PMAC acknowledges receipt of a valid <CR>-terminated command with an <ACK>; of an invalid command with a <BELL> character. Messages are sent as DATA <CR> [DATA <CR> ...] <ACK>. (The final <ACK> is the acknowledgment of the host command; it does not get sent with a message initiated from a PMAC program [SEND or CMD]). This is probably the best setting for fast communications with a host program without terminal display.
3: PMAC acknowledges receipt of a valid <CR>-terminated command with an <ACK>; of an invalid command with a <BELL> character. Messages are sent as <LF> DATA <CR> [<LF> DATA <CR> ...] <ACK>. (The final <ACK> is the acknowledgment of the host command; it does not get sent with a message initiated from a PMAC program [SEND or CMD]).

i4: Comm Integrity Mode This parameter permits PMAC to compute checksums of the communications bytes (characters) sent either way between the host and PMAC, and also controls how PMAC reacts to serial character errors (parity and framing), if found. Parity checking is only enabled if jumper E49 is removed.
The possible settings of I4 are:

Setting	Meaning
0	Checksum disabled, serial errors reported immediately
1	Checksum enabled, serial errors reported immediately
2	Checksum disabled, serial errors reported at end of line
3	Checksum enabled, serial errors reported at end of line

Communications Checksum: With I4=1 or 3, PMAC computes the checksum for communications in either direction and sends the checksum to the host. It is up to the host to do the comparison between PMAC's checksum and the checksum it computed itself. PMAC does not do this comparison. The host should

never send a checksum byte to PMAC.
Host-to-PMAC Checksum: PMAC will compute the checksum of a communications line sent from the host to PMAC. The checksum does not include any control characters sent (not even the final Carriage-Return). The checksum is sent to the host immediately following the acknowledging handshake character (<LF> or <ACK>), if any. Note that this acknowledging and handshake comes after any data response to the command (and its checksum!). If PMAC detects an error in the line through its normal syntax checking, it will respond with the <BELL> character, but will not follow this with a checksum byte.
PMAC-to-Host Checksum: PMAC will compute the checksum of any communications line it sends to the host. This checksum includes control characters sent with the line, including the final <carriage-return>. The checksum is sent immediately following this <carriage-return>. On a multiple-line response, one checksum is sent for each line. Note that this checksum is sent before the checksum of the command line that caused the response.
For more details on checksum, refer to the Writing a Host Communications Program section of the manual.
Serial character errors: If PMAC detects a serial character error, it will set a flag so that the entire command line will be rejected as having a syntax error after the <CR> is sent. With I4=0 or 1, it will also send a <BELL> character to the host immediately on detecting the character error. Note that this mode will catch a character error on the <CR> as well, whereas in the I4=2 or 3 mode, the host would have to catch an error on the <CR> character by the fact that PMAC would not respond (because it never saw a <CR>).

i6: Error Reporting Mode This parameter reports how PMAC reports errors in command lines. When I6 is set to 0 or 2, PMAC reports any error only with a <BELL> character. When I6 is 0, the <BELL> character is given for invalid commands issued both from the host and from PMAC programs (using CMD"{command}"). When I6 is 2, the <BELL> character is given only for invalid commands from the host; there is no response to invalid commands issued from PMAC programs. (In no mode is there a response to valid commands issued from PMAC programs.
When I6 is set to 1 or 3, an error number message can be reported along with the <BELL> character. The message comes in the form of ERRnnn<CR>, where nnn represents the three-digit error number. If I3 is set to 1 or 3, there is a <LF> character in front of the message.

When I6 is set to 1, the form of the error message is <BELL>{error message}. This setting is the best for interfacing with host-computer driver routines.
When I6 is set to 3, the form of the error message is <BELL><CR>{error message}. This setting is appropriate for use with the PMAC Executive Program in terminal mode.

i9: Program Listing Form Setting Meaning
0 Short form, decimal address I-variable return
1 Long form, decimal address I-variable return
2 Short form, hex address I-variable return
3 Long form, hex address I-variable return
When this parameter is 0 or 2, programs are sent back in abbreviated form for maximum compactness, and when I-variable values or M-variable definitions are requested, only the values or definitions are returned, not the full statements. When this parameter is 1 or 3, programs are sent back in full form for maximum readability. Also, I-variable values and M-variable definitions are returned as full command statements, which is useful for archiving and later downloading.
When this parameter is 0 or 1, I-variable values that specify PMAC addresses are returned in decimal form. When it is 2 or 3, these values are returned in hexadecimal form (with the '$' prefix). You are always free to send any I-variable values to PMAC either in hex or decimal, regardless of the I9 setting.

i56: DPRAM ASCII Comm Interrupt Enable

i58: DPRAM ASCII Comm Enable

i62: Internal Message <CR> Suppressed

PmacGlobalIVarMove

This cluster contains items for displaying and modifying PMAC's movement I-Variables.

Global Move I-Var Cluster This cluster contains items for displaying and modifying PMAC's movement I-Variables.

I32 **i11: Programmed Move Calc Time (mS)** This parameter controls the delay from when the run signal is taken (or the move sent if executing immediately) and when the first programmed move starts. If several PMAC's need to be run synchronously, I11 should be set the same on all of the cards. If I11 is set to zero, the first programmed move starts as soon as the calculation is complete. This calculation time delay is also used after any break in the continuous motion of a motion program : a DWELL, a PSET, a WAIT, or each move if Ix92=1 (a DELAY is technically a zero-distance move, and so does not constitute a break).
The actual delay time varies with the time base (e.g. at a % value of 50, the actual delay time will be twice the number defined here), which keeps it as a fixed distance of the master in an external time base application. If it is desired to have the slave coordinate system start up immediately with the master, I11 should be set to zero, and the program commanded to run before the master starts to move. If I11 is greater than zero, defining a definite time for calculations, and PMAC cannot complete the calculations for the first move by the end of the I11 time, PMAC will terminate the running of the program with a run-time error.

I32 **i12: Jog-to-Position Calc Time (mS)** This parameter controls how much time is allotted to calculate any jog move to a final position (J=, J={constant}, J/, J^{constant}, J:{constant}, or a motion program RAPID move). If a motor is currently jogging, it will continue on its present course during that time. If it is currently sitting still, it will continue to sit for this time.
This parameter should rarely need to be changed from the default. It should not be set to 0 for any reason, or PMAC will not be able to perform any of these types of moves. The minimum practical value for this parameter is 2 or 3.

I32 **i13: Programmed Move Segment Time (mS)** When greater than zero, this parameter puts PMAC into a mode where all moves are done as a continuous cubic spline in which the move segments are of the time length specified by the parameter in this variable (this is not the same thing as SPLINE mode moves). This mode is required for applications using CIRCLE mode moves. In this mode, all programs are sliced into even segments of I13 milliseconds, and the best cubic fit is applied for that segment, even across a move and/or dwell boundary.
Typical values of I13 are 5 to 10 msec. The smaller the value, the tighter the fit to the true curve, but the more computation is required for the moves, and the less is available for background tasks. If I13 is set

too low, PMAC will not be able to do all of its move calculations in the time allotted, and it will stop the motion program with a run-time error.
When I13=0, moves are done without this ongoing spline technique, and CIRCLE mode moves are done as LINEAR mode moves.

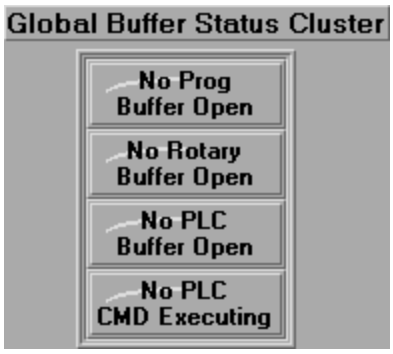
i14: Auto Position Match on Run This parameter, when set to 1, performs an automatic re-matching of motor and axis starting position registers to current motor commanded positions whenever a motion program is started. A mismatch can occur whenever a motor move (jog, open-loop, abort, or limit) changes the motor's target position without letting the axis position "know" of the change, or on power-up when an absolute position sensor starts up with a position other than zero.
With I14=1, PMAC will execute the PMATCH function on any Run or Step command to make sure that the axes in the motion program have the proper starting-position information. The only users who would not want this function, setting I14 to 0, are those who cannot afford the extra millisecond (approximately) of calculation time this requires.
With I14=0, PMAC uses the last motion program target position as the starting point for the calculations of the next move, even if these do not match the positions currently commanded for the motors assigned to the axes.

i50: Rapid Move Mode This parameter determines which variables are used for speed of RAPID mode moves. When I50 is set to 0, the jog parameter for each motor (Ix22) is used. When I50 is set to 1, the maximum velocity parameter for each motor (Ix16) is used instead. Regardless of the setting of I50, the jog acceleration parameters Ix19-Ix21 control the acceleration.

PmacGlobalStatBuffer

This is an indicator cluster for the global buffer status bits.

The colors of the indicators are indicative of the information conveyed by the particular bit. For example 'Prog Buffer Open' is Red when TRUE and grey otherwise.



Global Buffer Status Cluster This is an indicator cluster for the global buffer status bits

Program Buffer Open Status Bit 19 - Word 2 - Motion Buffer Open: Red if any motion program buffer (PROG or ROT) is open for entry. Grey if none of these buffers is open.

Rotary Buffer Open Status Bit 18 Word 2 - Rotary Buffer Open: Red if the rotary motion program buffer(s) (ROT) is (are) open for entry. Grey if this is (these are) closed.

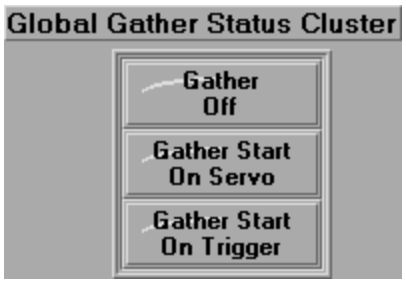
PLC Buffer Open Status Bit 17 Word 2 - PLC Buffer Open: Red if a PLC program buffer is open for entry. Grey if none of these buffers is open.

PLC CMD Executing Status Bit 16 Word 2 - PLC Command: Green if PMAC is processing a command issued from a PLC program. Grey otherwise. It is primarily for internal use.

PmacGlobalStatGather

This is an indicator cluster for the global gather status bits.

The colors of the indicators are indicative of the information conveyed by the particular bit. For example 'Gather Off' is Green when TRUE and grey otherwise.



Global Gather Status Cluster This is an indicator cluster for the global gather status bits

Gather On Status Bit 19 Word 1 - Data Gathering

Function On: Green when the data gathering function is active; Grey when the function is not active.

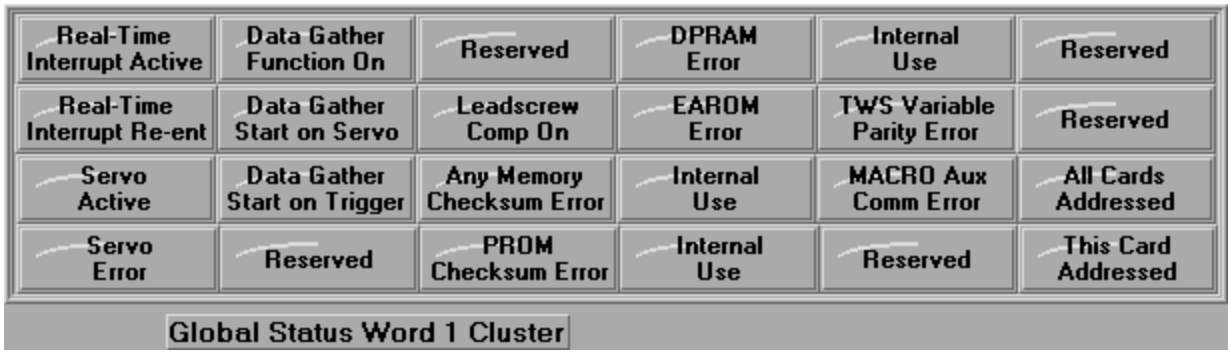
Gather Start on Servo Status Bit 18 Word 1 - Data Gather to Start on Servo: Green when the data gathering function is set up to start on the next servo cycle. Grey otherwise. It changes from Green to Grey as soon as the gathering function actually starts.

Gather Start on Trigger Status Bit 17 Word 1 - Data Gather to Start on Trigger: Green when the data gathering function is set up to start on the rising edge of Machine Input 2. Grey otherwise. It changes from Green to Grey as soon as the gathering function actually starts.

PmacGlobalStatWord1

This is an indicator cluster for the first global status word.

The colors of the indicators are indicative of the information conveyed by the particular bit. For example 'Servo Error' is Red when TRUE and grey otherwise. 'Data Gather Function On' is Green when TRUE and grey when FALSE.



Global Status Word 1 Cluster This is an indicator cluster for the first global status word

This Card Addressed

All Cards Addressed

Reserved Bit 2 (Reserved for Future Use)

Reserve Bit 3 (Reserved for Future Use)

Reserved Bit 4 (Reserved for Future Use)

MACRO Auxiliary Communication Error

TWS Variable Parity Error

Internal Use Bit 7 (Internal Use)

Internal Use Bit 8 (Internal Use)

Internal Use Bit 9 (Internal Use)

EAROM Error

DPRAM Error

PROM Checksum Error PROM Checksum Error: This bit is 1 if a firmware checksum error has been detected in PMAC's memory. It is 0 if a user program checksum error has been detected, or if no memory checksum error has been detected. Bit 13 distinguishes between these two cases.

Any Memory Checksum Error Status Bit 13 Word 1 - Any Memory Checksum Error: This bit is 1 if a checksum error has been detected for either the PMAC firmware or the user program buffer space. Bit 12 of this word distinguishes between the two cases.

Leadscrew Comp On Status Bit 14 Word 1 - Leadscrew Compensation On: This bit is 1 if leadscrew compensation is currently active in PMAC. It is 0 if the compensation is not active

Reserved Bit 15 Status Bit 15 Word 1 - Stimulus Function Active: This bit is 1 if the stimulus function is currently active in PMAC. It is 0 otherwise.

Reserved Bit 16 Status Bit 16 Word 1 - Stimulus Table Entered: This bit is 1 if a stimulus table has been entered into PMAC. It is 0 if there is no stimulus table in PMAC.

Data Gather Start on Trigger Status Bit 17 Word 1 - Data Gather to Start on Trigger: This bit is 1 when the data gathering function is set up to start on the rising edge of Machine Input 2. It is 0 otherwise. It changes from 1 to 0 as soon as the gathering function actually starts.

Data Gather Start on Servo Status Bit 17 Word 1 - Data Gather to Start on Servo: This bit is 1 when the data gathering function is set up to start on the next servo cycle. It is 0 otherwise. It changes from 1 to 0 as soon as the gathering function actually starts.

Data Gather Function On Status Bit 19 Word 1 - Data Gathering Function On: This bit is 1 when the data gathering function is active; it is 0 when the function is not active.

Servo Error Status Bit 20 Word 1 - Servo Error:

This bit is 1 if PMAC could not properly complete its servo routines. This is a serious error condition. It is 0 if the servo operations have been completing properly.

Servo Active Status Bit 21 Word 1 - Servo Active: This bit is 1 if PMAC is currently executing servo update operations. It is 0 if PMAC is executing other operations. Note that communications can only happen outside of the servo update, so polling this bit will always return a value of 0. This bit is for internal use.

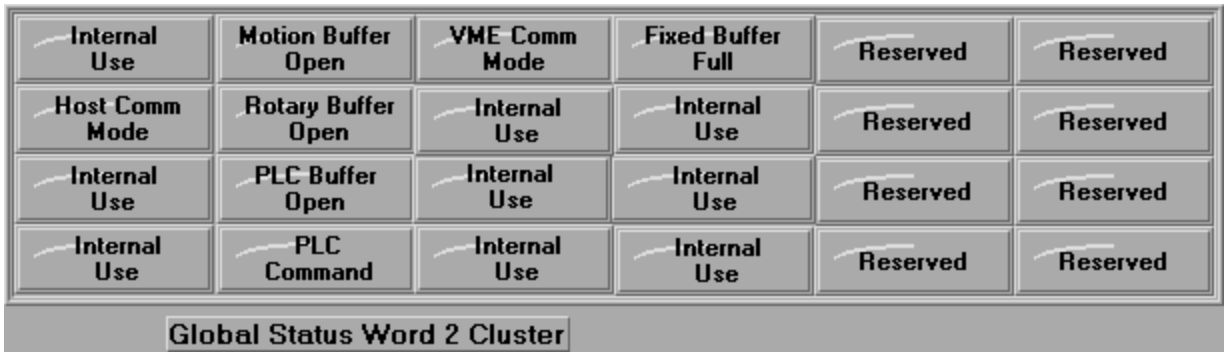
Real-Time Interrupt Re-entry Status Bit 22 Word 1 - Real-Time Interrupt Re-entry: This bit is 1 if a real-time interrupt task has taken long enough so that it was still executing when the next real-time interrupt came (I8+1 servo cycles later). It stays at 1 until the card is reset, or until this bit is manually changed to 0.

Real-Time Interrupt Active Status Bit 23 Word 1 - Real-Time Interrupt Active: This bit is 1 if PMAC is currently executing a real-time interrupt task (PLC 0 or motion program move planning). It is 0 if PMAC is executing some other task. Note: Communications can only happen outside of the real-time interrupt, so polling this bit will always return a value of 0. This bit is for internal use.

PmacGlobalStatWord2

This is an indicator cluster for the second global status word.

The colors of the indicators are indicative of the information conveyed by the particular bit. For example 'Motion Buffer Open' is Red when TRUE and grey otherwise. 'PLC Command' is Green when TRUE and grey when FALSE.



Global Status Word 2 Cluster This is an indicator cluster for the second global status word

Reserved Bit 0 (Reserved for future use)

Reserved Bit 1 (Reserved for future use)

Reserved Bit 2 (Reserved for future use)

Reserved Bit 3 (Reserved for future use)

Reserved Bit 4 (Reserved for future use)

Reserved Bit 5 (Reserved for future use)

Reserved Bit 6 (Reserved for future use)

Reserved Bit 7 (Reserved for future use)

Internal Use Bit 8 (Internal Use)

Internal Use Bit 9 (Internal Use)

Internal Use Bit 10 (Internal Use)

Fixed Buffer Full Status Bit 11 Word 2 - Fixed Buffer Full: This bit is 1 when no fixed motion (PROG) or PLC buffers are open, or when one is open but there are less than I18 words available. It is 0 when one of these buffers is open and there are more than I18 words available.

Internal Use Bit 12 (Internal Use)

Internal Use Bit 13 (Internal Use)

Internal Use Bit 14 (Internal Use)

VME Communication Mode Status Bit 15 Word 2 - VME Communication Mode: This bit is 1 when PMAC is prepared to send its communications over the VME bus "mailbox" port. It is 0 when PMAC is prepared to send its communications over the "host port" (PC bus or STD bus) or the serial port. It changes from 0 to 1 when it receives an alphanumeric command over the VME bus mailbox port. It changes from 1 to 0 when it receives a <CTRL-Z> over the serial port.

PLC Command Status Bit 16 Word 2 - PLC Command: This bit is 1 if PMAC is processing a command issued from a PLC program. It is 0 otherwise. It is primarily for internal use.

PLC Buffer Open Status Bit 17 Word 2 - PLC Buffer Open: This bit is 1 if a PLC program buffer is open for entry. It is 0 if none of these buffers is open.

Rotary Buffer Open Status Bit 18 Word 2 - Rotary

Buffer Open: This bit is 1 if the rotary motion program buffer(s) (ROT) is (are) open for entry. It is 0 if this is (these are) closed.

Motion Buffer Open Status Bit 19 Word 2 - Motion Buffer Open: This bit is 1 if any motion program buffer (PROG or ROT) is open for entry. It is 0 if none of these buffers is open.

Internal Use Bit 20 (Internal Use)

Internal Use Bit 21 (Internal Use)

Host Communication Mode Status Bit 22 Word 2 - Host Communication Mode: This bit is 1 when PMAC is prepared to send its communications over the "host port" (PC bus or STD bus). It is 0 when PMAC is prepared to send its communications over the VMEbus or the serial port. It changes from 0 to 1 when it receives an alphanumeric command over the host port. It changes from 1 to 0 when it receives a <CTRL-Z> over the serial port.

Internal Use Bit 23 (Internal Use)

PmacHome

\PmacHome – This collection of VIs provides access to the I-Variables and memory variables required for defining homing criteria and determining home position offset.

VIs

PmacHomeState

Query PMAC for important quantities used to determine the transformation of motor position encoder counts to actual encoder counts. These are presented as a Home State Cluster.

Device Number i32 (0) Provided by PmacDevOpen

Motor/Encoder Number i32 (1-8) (1) Motor to obtain home state information for. This assumes Motor 1 uses Encoder 1.

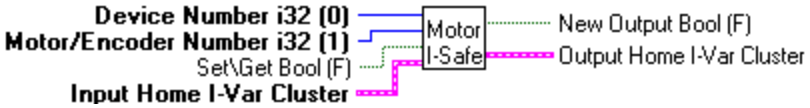
Get Home State Bool (F) When TRUE Home State Cluster contains the a variety of PMAC definitions defining the home state of the encoder and motor.

Home State Cluster This cluster contains several quantities used to determine the transformation of motor position

encoder counts to actual encoder counts.

PmacHomeIVar

Follow PMACView's standard I-Variable VI architecture. When Set/Get is TRUE the Home I-Variables for the specified Motor Encoder Number are set. Otherwise they are fetched from PMAC and provided by Output Home I-Var Cluster with New Output TRUE. Set/Get is not required and defaults to a Get operation.



Motor/Encoder Number i32 (1) Motor/Encoder number whose I-Variables are addressed. This assumes that encoder 1 is defined for motor 1, etc.

Device Number i32 (0) Provided by PmacDevOpen

Input Home I-Var Cluster This cluster contains items for displaying and modifying a motor/ encoder's home search motion and position capture trigger I-Variables.

Set\Get Bool (F) When TRUE PMAC is set using Input Home I-Var Cluster. When FALSE Output Home I-Var Cluster contains the current PMAC definitions.

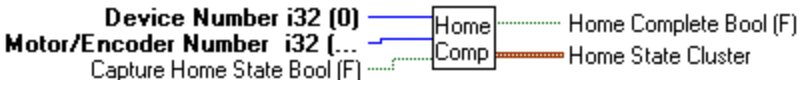
New Output Bool (F) TRUE when Output Home I-Var Cluster contains new data.

Output Home I-Var Cluster This cluster contains items for displaying and modifying a motor/ encoder's home search motion and position capture trigger I-Variables.

PmacHomeComplete

Create a PmacHomeState cluster containing I-Variables and memory registers for the specified Motor/Encoder number. The VI monitors the Home In Progress, Home Complete, and Desired Velocity Zero status bits for the motor to determine when to query PMAC for the required data. A query can also be forced if Capture Home State is TRUE.

This assumes Motor N uses Encoder N.



Device Number i32 (0) Provided by PmacDevOpen

Motor/Encoder Number i32 (1-8) (1) Motor to obtain home state information for. This assumes Motor 1 uses Encoder 1.

Capture Home State Bool (F) When TRUE force the

retrieval of the data for Home State Cluster.

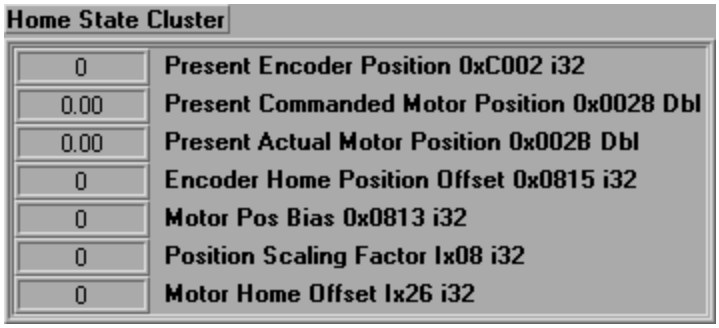
Home State Cluster This cluster contains several quantities used to determine the transformation of motor position encoder counts to actual encoder counts.

Home Complete Bool (F) Monitor Motor Number's status and process the Home status flags. When Home In-Progress is TRUE wait for Home Complete AND Desired Velocity Zero to be TRUE. Fetch the motor's Home State when these conditions are met.

Indicator and Control Clusters

PmacHomeState

This cluster contains several quantities used to determine the transformation of motor position encoder counts to actual encoder counts.



Home State Cluster This cluster contains several quantities used to determine the transformation of motor position encoder counts to actual encoder counts.

Present Encoder Position 0xC002 i32 The encoder Servo position register is 2 * Encoder counts with the LSB the direction bit. This output value is the actual encoder position referenced to a power-up/reset position of zero.

Present Commanded Motor Position 0x0028 Dbl This is the motor's present commanded position in units of 1 / (32 * Ix08) encoder counts referenced to the motor’s home position.

Present Actual Motor Position 0x002B Dbl This is the motor's present actual position in units of 1 / (32 * Ix08) encoder counts referenced to the motor’s home position.

Encoder Home Position Offset 0x0815 i32 This is the encoder's home offset position in encoder counts. It represents the difference between the encoder's

power-up/reset zero position and the position when a home operation completes.

I32 **Motor Pos Bias 0x0813 i32** This is the motor's position bias and represent the coordinate system translation in motor position encoder counts.

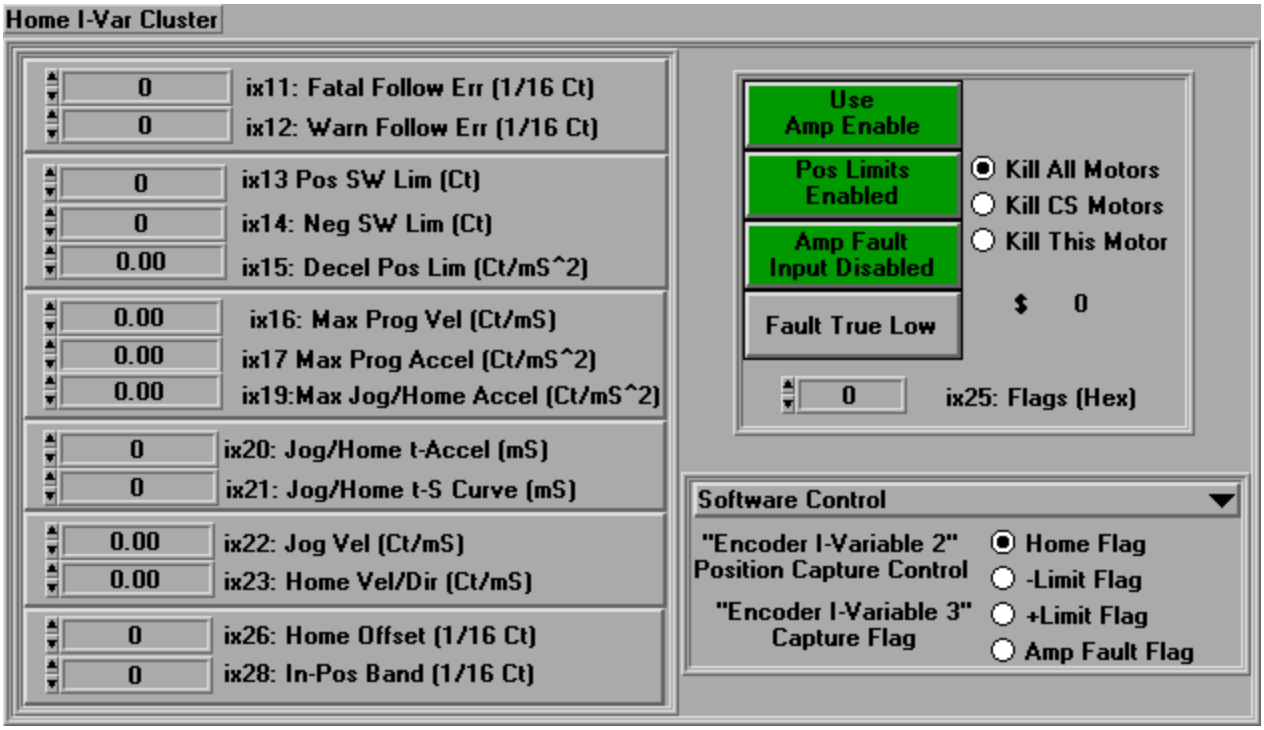
I32 **Position Scaling Factor Ix08 i32** This parameter controls how the position encoder counter gets extended into the full-length register. For most purposes, this is transparent to the user and does not need to be changed from the default.
There are two reasons that the user might want to change this from the default value. First, because it is involved in the "gear ratio" of the position following function -- the ratio is Ix07/Ix08 -- this might be changed (usually raised) to get a more precise ratio.
The second reason to change this parameter (usually lowering it) is to prevent internal saturation at very high gains or count rates (velocity). PMAC's filter will saturate when the velocity in counts/sec multiplied by Ix08 exceeds 256M (268,435,456). This only happens in very rare applications -- the count rate must exceed 2.8 million counts per second before the default value of Ix08 gives a problem.
When changing this parameter, make sure the motor is killed (disabled). Otherwise, a sudden jump will occur, because the internal position registers will have changed. This means that this parameter should not be changed in the middle of an application. If a real-time change in the position-following "gear ratio" is desired, Ix07 should be changed.
In most practical cases, Ix08 should not be set above 1000 because higher values can make the servo filter saturate too easily. If Ix08 is changed, Ix30 should be changed inversely to keep the same servo performance (e.g. if Ix08 is doubled, Ix30 should be halved).

I32 **Motor Home Offset Ix26 i32** This is the relative position of the end of the homing cycle to the position at which the home trigger was made. That is, the motor will command a stop at this distance from where it found the home flag(s), and call this commanded location as motor position zero.
This permits the motor zero position to be at a different location from the home trigger position, particularly useful when using an overtravel limit as a home flag (offsetting out of the limit before re-enabling the limit input as a limit). If large enough (greater than 1/2 times home speed times accel time) it permits a homing move without any reversal of direction.
The units of this parameter are 1/16 of a count, so the value should be 16 times the number of counts

between the trigger position and the home zero position.
Example:
If you wish your motor zero position to be 500 counts in the negative direction from the home trigger position, you would set Ix26 to -500 * 16 = -8000.

PmacHomeIVar

This cluster contains items and cluster for displaying and modifying a motors homing search and position capture trigger I-Variables.



Home I-Var Cluster This cluster contains items and cluster for displaying and modifying a motors homing search and position capture trigger I-Variables.

Motor Home Search I-Var Cluster This cluster contains items for displaying and modifying the most common motor homing search I-Variables.

ix11: Fatal Follow Err (1/16 Ct) This parameter sets the magnitude of the following error for motor x at which operation will shut down. When the magnitude of the following error exceeds Ix11, motor x is disabled (killed). If the motor's coordinate system is executing a program at the time, the program is aborted. It is optional whether other

PMAC motors are disabled when this motor exceeds its following error limit; bits 21 and 22 of Ix25 control what happens to the other motor (the default is that all PMAC motors are disabled).
A status bit for the motor, and one for the coordinate system (if the motor is in one) are set. If this coordinate system is hardware-selected on JPAN (with I2=0), or software-addressed by the host (with I2=1), the ERLD/ output on JPAN, and the EROR input to the interrupt controller (except for PMAC-VME) are triggered.
Setting Ix11 to zero disables the fatal following error limit for the motor. This may be desirable during initial development work, but it is strongly discouraged in an actual application. A fatal following error limit is a very important protection against various types of faults, such as loss of feedback, that cannot be detected directly, and that can cause severe damage to people and equipment.
The units of Ix11 are 1/16 of a count. Therefore this parameter must hold a value 16 times larger than the number of counts at which the limit will occur. For example, if the limit is to be 1000 counts, Ix11 should be set to 16,000.

ix12: Warn Follow Err (1/16 Ct) This parameter sets the magnitude of the following error for motor x at which a warning flag goes true. If this limit is exceeded, status bits are set for the motor and the motor's coordinate system (if any). The coordinate system status bit is the logical OR of the status bits of all the motors in the coordinate system.
Setting this parameter to zero disables the warning following error limit function. If this parameter is set greater than the fatal following error limit (Ix11), the warning status bit will never go true, because the fatal limit will disable the motor first.
At any given time, one coordinate system's status bit can be output to several places; which system depends on what coordinate system is hardware-selected on the panel input port if I2=0, or what coordinate system is software-addressed from the host (&n) if I2=1. The outputs that work in this way are F1LD/ (pin 23 on connector J2), F1ER (line IR3 into the programmable interrupt controller (PIC) on PMAC-PC,

line IR6 into the PIC on PMAC-STD) and, if E28 connects pins 1 and 2, FEFCO/ (on the JMACH connectors).
The units of Ix12 are 1/16 of a count. Therefore this parameter must hold a value 16 times larger than the number of counts at which the limit will occur. For example, if the limit is to be 1000 counts, Ix11 should be set to 16,000.

ix13 Pos SW Lim (Ct) This parameter sets the position for motor x which if exceeded in the positive direction causes a deceleration to a stop (controlled by Ix15) and allows no further positive position increments or positive output commands as long as the limit is exceeded. If this value is set to zero, there is no positive software limit (if you want 0 as a limit, use 1). This limit is automatically de-activated during homing search moves.
This limit is referenced to the most recent power-up zero position or homing move zero position. The physical position at which this limit occurs is not affected by axis offset commands (e.g. PSET, X=), although these commands will change the reported position value at which the limit occurs.

ix14: Neg SW Lim (Ct) This parameter sets the position for motor x which if exceeded in the negative direction causes a deceleration to a stop (controlled by Ix15) and allows no further negative position increments or negative output commands as long as the limit is exceeded. If this value is set to zero, there is no negative software limit (if you want 0 as a limit, use -1). This limit is automatically de-activated during homing search moves.
This limit is referenced to the most recent power-up zero position or homing move zero position. The physical position at which this limit occurs is not affected by axis offset commands (e.g. PSET, X=), although these commands will change the reported position value at which the limit occurs.

ix15: Decel Pos Lim (Ct/mS^2) This parameter sets the rate of deceleration that motor x will use if it exceeds a hardware or software limit, or has its motion aborted by command (A or <CONTROL-A>). This value should usually be set to a value near

the maximum physical capability of the motor. It is not a good idea to set this value past the capability of the motor, because doing so increases the likelihood of exceeding the following error limit, which stops the braking action, and could allow the axis to coast into a hard stop. Do not set this parameter to zero, or the motor will continue indefinitely after an abort or limit.
Suppose your motor had 125 encoder lines (500 counts) per millimeter, and you wished it to decelerate at 4000 mm/sec2. You would set Ix15 to 4000 mm/sec2 *500 cts/mm * sec2/1,000,000 msec2 = 2 cts/msec2.

ix16: Max Prog Vel (Ct/mS) This parameter sets a limit to the allowed velocity for LINEAR mode programmed moves for motor x, provided I13 equals zero (no move segmentation). If a blended move command in a motion program requests a higher velocity of this motor, all motors in the coordinate system are slowed down proportionately so that motor x will not exceed this parameter, yet the path will not be changed. This limit does not affect transition-point, circular, or splined moves. The calculation does not take into account any feedrate override (% value other than 100).
If PMAC's circular interpolation function is used at all, then I13 must be greater than zero, and Ix16 will no be active as a velocity limit.
This parameter also sets the speed of a programmed RAPID mode move for the motor, provided that variable I50 is set to 1 (if I50 is set to 0, jog speed parameter Ix22 is used instead). This happens regardless of the setting of I13.

ix17 Max Prog Accel (Ct/mS^2) This parameter sets a limit to the allowed acceleration in LINEAR-mode blended programmed moves for motor x, provided I13 equals zero (no move segmentation). If a LINEAR move command in a motion program requests a higher acceleration of this motor given its TA and TS time settings, the acceleration for all motors in the coordinate system is stretched out proportionately so that motor x will not exceed this parameter, yet the path will not be changed.

It is possible to have this limit govern the acceleration for all LINEAR-mode moves by setting very low TA and TS times. Do not set both the TA and TS times to zero, or a division-by-zero error will occur in the move calculations, possibly causing erratic movement. The minimum acceleration time settings that should be used are TA1 with TS0.
The default limit of 0.015625 counts/msec2 is quite low and will probably limit acceleration to a lower value than is desired in most systems; most users will eventually raise this limit This low default was used for safety reasons
When moves are broken into small pieces and blended together, this limit can affect the velocity, because it limits the calculated deceleration for each piece, even if that deceleration is never executed, because it blends into the next piece.
This limit does not affect PVT, CIRCLE, RAPID, or SPLINE moves. The calculation does not take into account any feedrate override (% value other than 100).
If PMAC's circular interpolation function is used at all, then I13 must be greater than zero, and Ix17 will no be active as an acceleration limit.

ix19:Max Jog/Home Accel (Ct/mS^2)
This parameter sets a limit to the commanded acceleration magnitude for jog and home moves, and for RAPID-mode programmed moves, of motor x. If the acceleration times in force at the time (Ix20 and Ix21) request a higher rate of acceleration, this rate of acceleration will be used instead. The calculation does not take into account any feedrate override (% value other than 100).
Since jogging moves are usually not coordinated between motors, many people prefer to specify jog acceleration by rate, not time. To do this, simply set Ix20 and Ix21 low enough that the Ix19 limit is always used. Do not set both Ix20 and Ix21 to 0; or a division-by-zero error will result in the move calculations, possibly causing erratic operations. The minimum acceleration time settings that should be used are Ix20=1 and Ix21=0.
The default limit of 0.015625 counts/msec2 is quite low and will probably limit acceleration to a lower value than is desired in most systems; most users

will eventually raise this limit This low default was used for safety reasons.
Examples
With Ix20 (accel time) at 100 msec, Ix21 (S-curve time) at 0, and Ix22 (jog speed) at 50 counts/msec, a jog command from stop would request an acceleration of (50 cts/msec) / 100 msec, or 0.5 cts/msec2. If Ix19 were set to 0.25, the acceleration would be done in 200 msec, not 100 msec. With the same parameters in force, an on-the-fly reversal from positive to negative jog would request an acceleration of (50-(-50) cts/msec) / 100 msec, or 1.0 cts/msec2. The limit would extend this acceleration by a factor of 4, to 400 msec.

ix20: Jog/Home t-Accel (mS) This parameter establishes the time spent in acceleration in a jogging, homing, or programmed RAPID-mode move (starting, stopping, and changing speeds). However, if Ix21 (jog/home S-curve time) is greater than half this parameter, the total time spent in acceleration will be 2 times Ix21. Therefore, if Ix20 is set to 0, Ix21 alone controls the acceleration time in "pure" S-curve form. In addition, if the maximum acceleration rate set by these times exceeds what is permitted for the motor (Ix19), the time will be increased so that Ix19 is not exceeded.
Do not set both Ix20 and Ix21 to 0 simultaneously, even if you are relying on Ix19 to limit your acceleration, or a division-by-zero error will occur in the jog move calculations, possibly resulting in erratic motion.
A change in this parameter will not take effect until the next move command. For instance, if you wanted a different deceleration time from acceleration time in a jog move, you would specify the acceleration time, command the jog, change the deceleration time, then command the jog move again (e.g. J=), or at least the end of the jog (J/).

ix21: Jog/Home t-S Curve (mS) This parameter establishes the time spent in each "half" of the "S" for S-curve acceleration in a jogging, homing, or RAPID-mode move (starting, stopping, and changing speeds). If this parameter is more than half of Ix20, the total acceleration time will be 2 times Ix21, and

the acceleration time will be "pure" S-curve (no constant acceleration portion). If the maximum acceleration rate set by Ix20 and Ix21 exceeds what is permitted for the motor (Ix19), the time will be increased so that Ix19 is not exceeded.
Do not set both Ix20 and Ix21 to 0 simultaneously, even if you are relying on Ix19 to limit your acceleration, or a division-by-zero error will occur in the jog move calculations, possibly resulting in erratic motion.
A change in this parameter will not take effect until the next move command. For instance, if you wanted a different deceleration time from acceleration time in a jog move, you would specify the acceleration time, command the jog, change the deceleration time, then command the jog move again (e.g. J=), or at least the end of the jog (J/)

ix22: Jog Vel (Ct/mS) This parameter establishes the commanded speed of a jog move, or a programmed RAPID-mode move (if I50=0) for motor x. Direction of the jog move is controlled by the jog command.
A change in this parameter will not take effect until the next move command. For instance, if you wanted to change the jog speed on the fly, you would start the jog move, change this parameter, then issue a new jog command.

ix23: Home Vel/Dir (Ct/mS) This parameter establishes the commanded speed and direction of a homing-search move for motor x. Changing the sign reverses the direction of the homing move -- a negative value specifies a home search in the negative direction; a positive value specifies the positive direction.

ix26: Home Offset (1/16 Ct) This is the relative position of the end of the homing cycle to the position at which the home trigger was made. That is, the motor will command a stop at this distance from where it found the home flag(s), and call this commanded location as motor position zero.
This permits the motor zero position to be at a different location from the home trigger position, particularly useful when using an overtravel limit as a home flag

(offsetting out of the limit before re-enabling the limit input as a limit). If large enough (greater than 1/2 times home speed times accel time) it permits a homing move without any reversal of direction.

ix28: In-Pos Band (1/16 Ct) This is the magnitude of the maximum following error at which motor x will be considered "in position" when not performing a move. Several things happen when the motor is "in-position". First, a status bit in the motor status word is set. Second, if all other motors in the same coordinate system are also "in-position", a status bit in the coordinate system status word is set. Third, for the hardware-selected (FPD0/-FPD3/) coordinate system -- if I2=0 -- or for the software addressed (&n) coordinate system -- if I2=1 -- outputs to the control panel port and to the interrupt controller are set.
Technically, four conditions must be met for a motor to be considered "in-position":
1.) the motor must be in closed-loop control
2.) the desired velocity must be zero;
3.) the magnitude of the following error must be less than this parameter;
4.) the move timer must not be active.
The move timer is active during any programmed or non-programmed move, including DWELLs and DELAYs in a program -- if you wish this bit to come true during a program, you must do an indefinite wait between some moves by keeping the program trapped in a WHILE loop that has no moves or DWELLs. More sophisticated in-position functions (for instance, ones that require several consecutive scans within the band) can be implemented using PLC programs. See the program examples section.
The units of this parameter are 1/16 of a count, so the value should be 16 times the number of counts in the in-position band.

Motor Flag I-Var Cluster This cluster contains items for displaying and modifying the ix25 - motor encoder flag I-Variables. This parameter tells PMAC what set of inputs it will look to for motor x's overtravel limit switches, home flag, amplifier-fault flag, and amplifier-enable output. Typically, these are the flags associated with an encoder input; specifically, those of the position feedback encoder for the motor.

Use Amplifier Enable Amplifier Enable
Use Bit: With bit 16 equal to zero -- the normal case -- the AENAn/DIRn output is used as an amplifier-enable line: off when the motor is "killed", on when it is enabled. Voltage polarity is determined by jumper(s) E17.
If bit 16 (value $10000, or 65536) is set to one (e.g. I125=$1C000), this output is not used as an amplifier-enable line. This permits use of the line as a direction bit for applications requiring magnitude-and direction outputs, such as driving steppers through
voltage-to-frequency converters. (Setting bit 16 of Ix02 to 1 enables use of this output as a direction bit.) General-purpose use of this output is also possible by assigning an M-variable to it.

Enable Position Limits Overtravel Limit
Use Bit: With bit 17 equal to zero -- the normal case -- the +/-LIMn inputs must be held low to permit commanded motion in the appropriate direction. If there are not actual (normally closed or normally conducting) limit switches, the inputs must be hardwired to ground.
The direction sense of the limit inputs is the opposite of what many people consider intuitive. That is, the +LIMn input, when taken high (opened), stops commanded motion in the negative direction; the -LIMn input, when taken high, stops commanded motion in the positive direction. It is important to confirm the direction sense of your limit inputs in actual operation.
If bit 17 (value $20000, or 131072) is set to one (e.g. I125=$2C000), motor x does not use these inputs as overtravel limits. This can be done temporarily, as when using a limit as a homing flag. If the limit function is not used at all, these inputs can be used as general-purpose inputs by assigning M-variables to them.

Enable Amplifier Fault Amplifier Fault
Use Bit: If bit 20 of Ix25 is 0, the amplifier-fault input function through the FAULTn input is enabled. If bit 20 (value $100000, or 1,048,576) is 1 (e.g. I125=$10C000), this function is disabled. General-purpose use of this input is then possible by assigning an M-variable to the input.

Fault Input Level Amplifier-Fault Polarity Bit: Bit 23 (value 8,388,608) of Ix25 controls the polarity of the amplifier fault input. A zero in this bit means a low-true input (low means fault); a one means high-true (high means fault). The input is pulled high internally, so if no line is attached to the input, and bit 20 of Ix25 is zero (enabling the fault function), bit 23 of Ix25 must be zero to permit operation of the motor.

On Fault Action-on-Fault Bits: Bits 21 (value $200000, or 2,097,152) and 22 (value $400000, or 4,194,344) of Ix25 control what action is taken on an amplifier fault for the motor, or on exceeding the fatal following error limit (Ix11) for the motor:

Bit 22	Bit 21	Functions
Bit 22=0	Bit 21=0:	Kill all PMAC motors
Bit 22=0	Bit 21=1:	Kill all motors in same coordinate system
Bit 22=1	Bit 21=0:	Kill only this motor
Bit 22=1	Bit 21=1:	Kill only this motor

Regardless of the setting of these bits, a program running in the coordinate system of the offending motor will be halted on an amplifier fault or the exceeding of a fatal following error limit.

Flag Address Variable Hex Decimal Limit and Flag set

I125	$C000	(49152)	(LIM1, HMFL1...)
I225	$C004	(49156)	(LIM2, HMFL2...)
I325	$C008	(49160)	(LIM3, HMFL3...)
I425	$C00C	(49164)	(LIM4, HMFL4...)
I525	$C010	(49168)	(LIM5, HMFL5...)
I625	$C014	(49172)	(LIM6, HMFL6...)
I725	$C018	(49176)	(LIM7, HMFL7...)
I825	$C01C	(49180)	(LIM8, HMFL8...)

The addresses for the first eight sets of inputs are given in the default table, above. The addresses for input sets 9 through 16,

which are on the Accessory 24 Axis Expansion Board, are given below:

Limit and Flag set	Hex	Decimal
LIM9, HMFL9	$C020	(49184)
LIM10, HMFL10	$C024	(49188)
LIM11, HMFL11	$C028	(49192)
LIM12, HMFL12	$C02C	(49196)
LIM13, HMFL13	$C030	(49200)
LIM14, HMFL14	$C034	(49204)
LIM15, HMFL15	$C038	(49208)
LIM16, HMFL16	$C03C	(49212)

The overtravel-limit inputs specified by this parameter must be held low in order for motor x to be able to command movement. The polarity of the amplifier-fault input is determined by a high-order bit of this parameter (see below). The polarity of the home-flag input is determined by the Encoder/Flag I-Variables 2 and 3 for the specified encoder. The polarity of the amplifier-enable output is determined by Jumper E17.

Extended Addressing: The source address of the flag information occupies bits 0 to 15 of Ix25 (range $0000 to $FFFF, or 0 to 65535). If this is all that is specified -- that is, all higher bits are zero -- then all of the flags are used, and used in the "normal" mode (low-true FAULT, disabling all motors). If higher bits are set to one, some of the flags are not used, or used in an alternate manner, as documented below.

ix25: Flags (Hex) This is the hex value of Ix25.

This parameter tells PMAC what set of inputs it will look to for motor x's overtravel limit switches, home flag, amplifier-fault flag, and amplifier-enable output. Typically, these are the flags associated with an encoder input; specifically, those of the position feedback encoder for the motor.

Encoder Capture I-Var Cluster This cluster contains items for displaying and modifying Encoder I-Variables 2 and 3. These parameters tell PMAC what trigger conditions and flags to use for position capture.

"Encoder I-Variable 2" Position Capture Control This parameter determines which signal or combination of signals (and which polarity) triggers a position capture of the counter for encoder

n. If a flag input (home, limit, or fault) is used, I903 (etc.) determines which flag. Proper setup of this variable is essential for a successful home search, which depends on the position-capture function. The following settings may be used:

Setting	Meaning
0	Software Control
1	Rising edge of CHCn (third channel)
2	Rising edge of Flag n (as set by Flag Select)
3	Rising edge of [CHCn AND Flag n]
4	Software Control
5	Falling edge of CHCn (third channel)
6	Rising edge of Flag n (as set by Flag Select)
7	Rising edge of [CHCn/ AND Flag n]
8	Software Control
9	Rising edge of CHCn (third channel)
10	Falling edge of Flag n (as set by Flag Select)
11	Rising edge of [CHCn AND Flag n/]
12	Software Control
13	Falling edge of CHCn (third channel)
14	Falling edge of Flag n (as set by Flag Select)
15	Rising edge of [CHCn/ AND Flag n/]

Note that several of these values are redundant. To do a software-controlled position capture, preset this parameter to 0 or 4; when the parameter is then changed to 8 or 12, the capture is triggered (this is not of much practical use).

"Encoder I-Variable 3" Capture Flag

This parameter determines which of the "Flag" inputs will be used for position capture (if one is used -- see I902 etc.):

Setting	Meaning
0	HMFLn (Home Flag n)
1	-LIMn (Negative Limit Signal n)
2	+LIMn (Positive Limit Signal n)
3	FAULTn (Amplifier Fault Signal n)

This parameter is typically set to zero, because in actual use, the +/-LIMn and

FAULTn flags create other effects that usually interfere with what is trying to be accomplished by the position capture. If you wish to capture on the +/-LIMn or FAULTn flags, you must either disable their normal functions with Ix25, or use a channel n where none of the flags is used for the normal axis functions.

PmacIVar

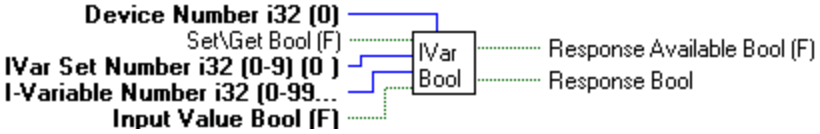
\PmacIVar - Provide direct binary access to PMAC I-Variables. This collection of VIs provides an easy to use architecture for accessing I-Variables and avoids formatting problems that can arise when querying PMAC I-Variables that might be returned as hexadecimal values.

VIs

PmacIVarBool

If Set\Get is FALSE or not wired get the Boolean I-Variable specified by IVar Set Number and I-Variable Number. Response Available will be TRUE to indicate Response contains the new value. If Set\Get is TRUE set the Boolean I-Variable using Input Value. Response Available will be FALSE and Response defaults to Input Value.

The variable address is calculated as IVar Set Number * 100 + I-Variable Number. IVar Set Number = 0 addresses global I-Variables. IVar Set Numbers from 1 - 8 address motors and coordinate system I-Variables.



Device Number i32 (0) DeviceNumber

I-Variable Number i32 (0-99) (0) I-Variable offset within IVar Set

Input Value Bool (F) Input Value for Set operation

IVar Set Number i32 (0-9) (0) 100's value of I-Variable to access

Set\Get Bool (F) Set I-Variable when TRUE. Get I-Variable when FALSE or unwired.

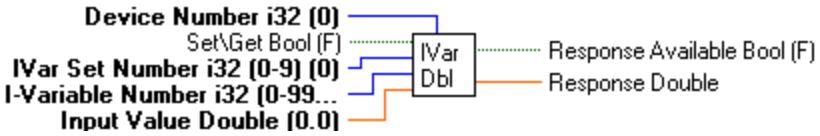
Response Bool Value of I-Variable for Get operation. Input Value for Set Operation.

Response Available Bool (F) TRUE when Get operation produces valid Response

PmacIVarDbl

If Set\Get is FALSE or not wired get the Double I-Variable specified by IVar Set Number and I-Variable Number. Response Available will be TRUE to indicate Response contains the new value. If Set\Get is TRUE set the Double I-Variable using Input Value. Response Available will be FALSE and Response defaults to Input Value.

The variable address is calculated as IVar Set Number * 100 + I-Variable Number. IVar Set Number = 0 addresses global I-Variables. IVar Set Numbers from 1 - 8 address motors and coordinate system I-Variables.



Device Number i32 (0) Provided by PmacDevOpen

I-Variable Number i32 (0-99) (0) I-Variable offset within IVar Set

Input Value Double (0.0) Input Value for Set operation

IVar Set Number i32 (0-9) (0) 100's value of I-Variable to access

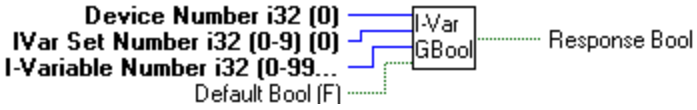
Set\Get Bool (F) Set I-Variable when TRUE. Get I-Variable when FALSE or unwired.

Response Double Value of I-Variable for Get operation. Input Value for Set Operation.

Response Available Bool (F) TRUE when Get operation produces valid Response

PmacIVarGetBool

Get the Boolean I-Variable specified by IVar Set Number and I-Variable Number. The variable address is calculated as IVar Set Number * 100 + I-Variable Number. IVar Set Number = 0 addresses global I-Variables. IVar Set Numbers from 1 - 8 address motors and coordinate system I-Variables.



Device Number i32 (0) Provided by PmacDevOpen

I-Variable Number i32 (0-99) (0) I-Variable offset within IVar Set

Default Bool (F) Default value

IVar Set Number i32 (0-9) (0) 100's value of I-Variable to access

Response Bool I-Variable value

PmacIVarGetDbl

Get the Double I-Variable specified by IVar Set Number and I-Variable Number. The variable address is calculated as IVar Set Number * 100 + I-Variable Number. IVar Set Number = 0 addresses global I-Variables. IVar Set Numbers from 1 - 8 address motors and coordinate system I-Variables.

Device Number i32 (0) Provided by PmacDevOpen

I-Variable Number i32 (0-99) (0) I-Variable offset within IVar Set

Default Double (0.0) Default

IVar Set Number i32 (0-9) (0) 100's value of I-Variable to access

Response Double I-Variable value

PmacIVarGetLong

Get the Long I-Variable specified by IVar Set Number and I-Variable Number. The variable address is calculated as IVar Set Number * 100 + I-Variable Number. IVar Set Number = 0 addresses global I-Variables. IVar Set Numbers from 1 - 8 address motors and coordinate system I-Variables.

Device Number i32 (0) Provided by PmacDevOpen

I-Variable Number i32 (0-99) (0) I-Variable offset within IVar Set

Default i32 (0) Default value

IVar Set Number i32 (0-9) (0) 100's value of I-Variable to access

Response i32 I-Variable value

PmacIVarGetShort

Get the Short I-Variable specified by IVar Set Number and I-Variable Number. The variable address is calculated as IVar Set Number * 100 + I-Variable Number. IVar Set Number = 0 addresses global I-Variables. IVar Set Numbers from 1 - 8 address motors and coordinate system I-Variables.

Device Number i32 (0) Provided by PmacDevOpen

I-Variable Number i32 (0-99) (0) I-Variable offset within IVar Set

Default i16 (0) Default value

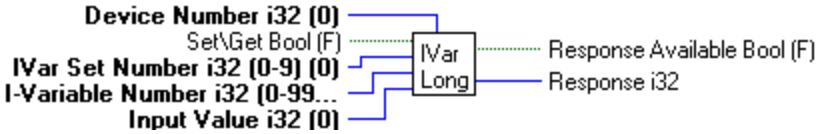
IVar Set Number i32 (0-9) (0) 100's value of I-Variable to access

Response i16 I-Variable value

PmacIVarLong

If Set\Get is FALSE or not wired get the Long I-Variable specified by IVar Set Number and I-Variable Number. Response Available will be TRUE to indicate Response contains the new value. If Set\Get is TRUE set the Long I-Variable using Input Value. Response Available will be FALSE and Response defaults to Input Value.

The variable address is calculated as IVar Set Number * 100 + I-Variable Number. IVar Set Number = 0 addresses global I-Variables. IVar Set Numbers from 1 - 8 address motors and coordinate system I-Variables.



Device Number i32 (0) Provided by PmacDevOpen

I-Variable Number i32 (0-99) (0) I-Variable offset within IVar Set

Input Value i32 (0) Input Value for Set operation

IVar Set Number i32 (0-9) (0) 100's value of I-Variable to access

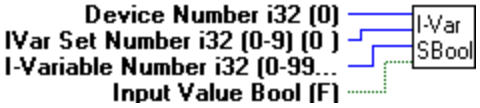
Set\Get Bool (F) Set I-Variable when TRUE. Get I-Variable when FALSE or unwired.

Response i32 Value of I-Variable for Get operation. Input Value for Set Operation.

Response Available Bool (F) TRUE when Get operation produces valid Response

PmacIVarSetBool

Set the Boolean I-Variable specified by IVar Set Number and I-Variable Number. The variable address is calculated as IVar Set Number * 100 + I-Variable Number. IVar Set Number = 0 addresses global I-Variables. IVar Set Numbers from 1 - 8 address motors and coordinate system I-Variables.



Device Number i32 (0) Provided by PmacDevOpen

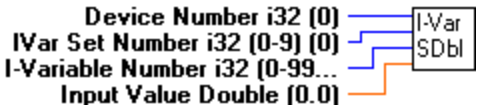
I-Variable Number i32 (0-99) (0) I-Variable offset within IVar Set

Input Value Bool (F) Input Value for specified I-Variable

IVar Set Number i32 (0-9) (0) 100's value of I-Variable to access

PmacIVarSetDbl

Set the Double I-Variable specified by IVar Set Number and I-Variable Number. The variable address is calculated as IVar Set Number * 100 + I-Variable Number. IVar Set Number = 0 addresses global I-Variables. IVar Set Numbers from 1 - 8 address motors and coordinate system I-Variables.



Device Number i32 (0) DeviceNumber

I-Variable Number i32 (0-99) (0) I-Variable offset within IVar Set

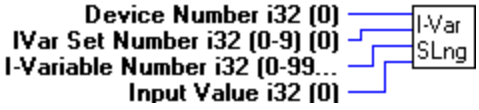
Input Value Double (0.0) Input Value for specified I-Variable

IVar Set Number i32 (0-9) (0) 100's value of I-Variable to access

PmacIVarSetLong

Set the Long I-Variable specified by IVar Set Number and I-Variable Number. The variable address is calculated as IVar Set Number * 100 + I-Variable Number. IVar

Set Number = 0 addresses global I-Variables. IVar Set Numbers from 1 - 8 address motors and coordinate system I-Variables.



Device Number i32 (0) Provided by PmacDevOpen

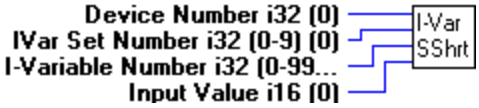
I-Variable Number i32 (0-99) (0) I-Variable offset within IVar Set

Input Value i32 (0) Input Value for specified I-Variable

IVar Set Number i32 (0-9) (0) 100's value of I-Variable to access

PmacIVarSetShort

Set the Short I-Variable specified by IVar Set Number and I-Variable Number. The variable address is calculated as IVar Set Number * 100 + I-Variable Number. IVar Set Number = 0 addresses global I-Variables. IVar Set Numbers from 1 - 8 address motors and coordinate system I-Variables.



Device Number i32 (0) Provided by PmacDevOpen

I-Variable Number i32 (0-99) (0) I-Variable offset within IVar Set

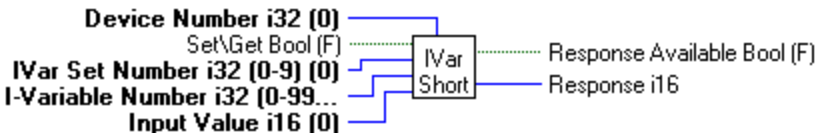
Input Value i16 (0) Input Value for specified I-Variable

IVar Set Number i32 (0-9) (0) 100's value of I-Variable to access

PmacIVarShort

If Set\Get is FALSE or not wired get the Short I-Variable specified by IVar Set Number and I-Variable Number. Response Available will be TRUE to indicate Response contains the new value. If Set\Get is TRUE set the Short I-Variable using Input Value. Response Available will be FALSE and Response defaults to Input Value.

The variable address is calculated as IVar Set Number * 100 + I-Variable Number. IVar Set Number = 0 addresses global I-Variables. IVar Set Numbers from 1 - 8 address motors and coordinate system I-Variables.



I32 **Device Number i32 (0)** Provided by PmacDevOpen

I32 **I-Variable Number i32 (0-99) (0)** I-Variable offset within IVar Set

I16 **Input Value i16 (0)** Input Value for Set operation

I32 **IVar Set Number i32 (0-9) (0)** 100's value of I-Variable to access

TF **Set\Get Bool (F)** Set I-Variable when TRUE. Get I-Variable when FALSE or unwired.

I16 **Response i16** Value of I-Variable for Get operation. Input Value for Set Operation.

TF **Response Available Bool (F)** TRUE when Get operation produces valid Response

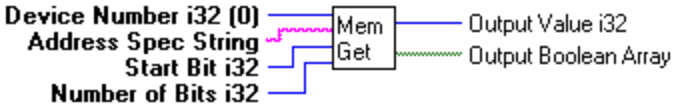
PmacMemory

\PmacMemory- Access to PMAC's memory mapped variables and registers are simplified with this group of VIs.

VIs

PmacMemoryGet

Output Value is the value of the bit field defined by Start Bit and Number of Bits at the specified memory address. Output the field as both Output Value and Output Boolean Array.



abc **Address Spec String** Address to access. e.g. X:$002B

I32 **Start Bit i32** Start of bit field

I32 **Number of Bits i32** Number of bits in bit field

I32 **Device Number i32 (0)** Provided by PmacDevOpen

I32 **Output Value i32** Value of bits in field

[TF] **Output Boolean Array** Boolean array representation of bit

field

PmacMemoryGetBit

Bit Value is the bit at Bit Number in Input Value.

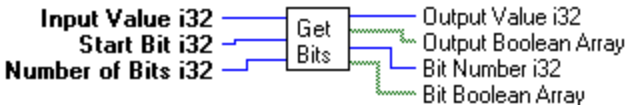
Input Value i32 Word to read a bit from

Bit Number i32 Bit number to be read

Bit Value Bool Bit value from Input Value

PmacMemoryGetBits

Fetch the field defined by Start Bit and Number of Bits from Input Value. Return the field as Output Value i32, Output Boolean Array, and Bit Number (same as Output Value). Bit Boolean Array can be used with sets of radio buttons. If Output Value = 3 then Bit Boolean Array is 0, 0, 0, 1.



Number of Bits i32 Number of bits in field

Start Bit i32 Start bit of field to fetch

Input Value i32 Word to read a bit field from

Output Value i32 Value of bit field

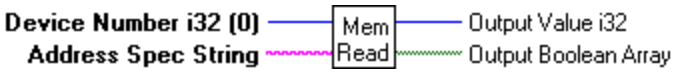
Output Boolean Array Boolean array representation of bit field

Bit Number i32 Same as Output Value.

Bit Boolean Array 2^n Boolean array representation of field. If field value is 3 this array is 0, 0, 0, 1.

PmacMemoryRead

Read a 24 bit quantity from the memory location specified by Address Spec String. For example, X:$002B. The result is output as both an i32 and a Boolean array.



Address Spec String Address to access. e.g. X:$002B

Device Number i32 (0) Provided by PmacDevOpen

Output Value i32 Value read from memory

Output Boolean Array Boolean array representation of data read from memory.

PmacMemoryReadDbl

Read a 48 bit quantity from the memory location specified by Address Spec String. For example, L:$002B. The result is output as both a double and a hi-word and lo-word.

Address Spec String Address to access. e.g. X:$002B

Device Number i32 (0) Provided by PmacDevOpen

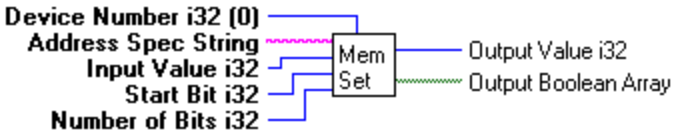
Output Value i32 48 bit data read from memory

Output Hi Word i32 hi-word of 48 bits read from memory

Output Lo Word i32 lo-word of 48 bit data read from memory

PmacMemorySet

Write Input Value to a bit field defined by Start Bit and Number of Bits at the specified memory address. Output Value and Output Boolean Array are the value of the entire memory location with the new field.



Number of Bits i32 Number of bits in bit field

Input Value i32 Value to write to bit field

Start Bit i32 Start of bit field

Address Spec String Address to access. e.g. X:$002B

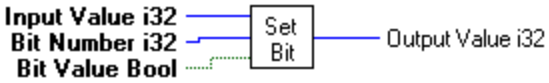
Device Number i32 (0) Provided by PmacDevOpen

Output Boolean Array Boolean array representation of bit field

Output Value i32 Value of bits in field

PmacMemorySetBit

Set Bit Number in Input Value using Bit Value. The new word is Output Value.



Bit Value Bool Value of bit

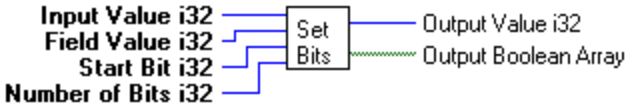
Input Value i32 Word in which to set bit value

Bit Number i32 Bit number to set

Output Value i32

PmacMemorySetBits

Insert Field Value into Input Value at the field defined by Start Bit and Number of Bits. Output Value is Input Value with Field Value inserted. Output Boolean Array is the Boolean representation of Output Value.



Number of Bits i32 Number of bits in field

Start Bit i32 Start bit of field

Field Value i32 Value to insert into field

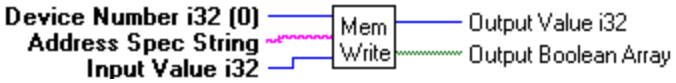
Input Value i32 Word in which field will be set

Output Value i32 Input Value with Field Value inserted

Boolean Array Boolean array representation of Input Value with Field Value inserted.

PmacMemoryWrite

Write a 24 bit quantity (Input Value) to the memory location specified by Address Spec String. For example, X:$002B. Pass Input Value to Output Value and Output Boolean Array.



Input Value i32 integer value to write to the specified address

Address Spec String Address to access. e.g. X:$002B

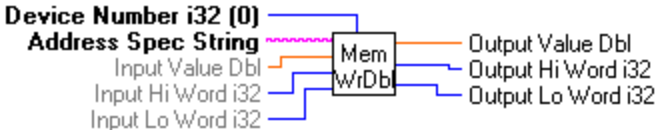
Device Number i32 (0) Provided by PmacDevOpen

Output Boolean Array Boolean array representation of Input Value written to memory

Output Value i32 Copy of Input Value written to memory

PmacMemoryWriteDbl

Write a 48 bit quantity (Input Value) to the memory location specified by Address Spec String. For example, L:$002B. Input Value is copied to Output Value.



Input Value Dbl double value to write to the specified address

Address Spec String Address to access. e.g. X:$002B

Device Number i32 (0) Provided by PmacDevOpen

Input Hi Word i32 hi-word to build 48 bit word to write

Input Lo Word i32 low word to build 48 bit word to write

Output Value Dbl Copy of Input Value written to memory

Output Hi Word i32 hi-word of 48 bit data written to memory

Output Lo Word i32 lo-word of 48 bit data written to memory

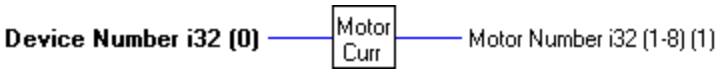
PmacMotor

\PmacMotor - Monitor and control individual motors. There are numerous ICVs to get motor position, velocity, and following error, modify motor I-Variables, process motor status, and jog motors.

VIs

PmacMotorCurrent

Query PMAC for the currently addressed motor. It is most generally used in interactive development environments rather than a custom VI where you want to address a specific motor.



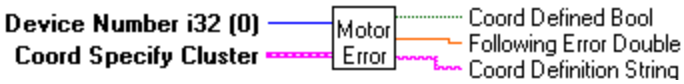
Device Number i32 (0) Provided by PmacDevOpen

Motor Number i32 (1-8) (1) Currently addressed motor number in PMAC

PmacMotorError

Query PMAC for Motor Number's following error. Following error is the difference between motor desired and measured position at any instant. When the motor is open-loop (killed or enabled), following error does not exist and PMAC reports a value of 0.

Coord Specify Cluster specifies a motor within a CS and an attempt to convert motor following error from encoder counts to CS units. If the motor is not defined in the CS no conversion is applied. If the motor is defined and Convert is TRUE Coord Defined is TRUE and following error is scaled from encoder counts to CS units. Coord Definition is a string specifying following error units as "Encoder" or the CS definition of the motor.



Device Number i32 (0) Provided by PmacDevOpen

Coord Specify Cluster Cluster defining the motor, CS, and conversion state to be applied

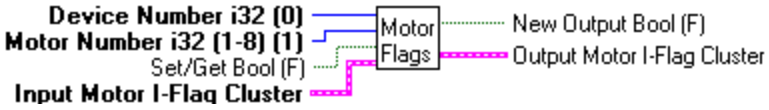
Coord Defined Bool If Motor Number is defined in Coord Number and Convert Bool is TRUE this is TRUE.

Coord Definition String String defining the motor within the CS. "Encoder" if not defined.

Following Error Double This value is the difference between motor desired and measured position at any instant. When the motor is open-loop (killed or enabled), following error does not exist and PMAC reports a value of 0
Following Error is in CS units if Convert is TRUE and the motor is defined in the CS. Otherwise motor following error in encoder counts..

PmacMotorIVarFlag

Follow PMACPanel's standard I-Variable VI architecture. When Set/Get is TRUE the encoder flag I-Variable Ix25 for the specified Coord Number are set. Otherwise they are fetched from PMAC and provided by Output Coord I-Flag Cluster with New Output TRUE. Set/Get is not required and defaults to a Get operation.



Device Number i32 (0) Provided by PmacDevOpen

Motor Number i32 (1-8) (1) Motor number whose I-Variables are addressed

Set/Get Bool (F) When TRUE PMAC is set using Input Motor I-Flag Cluster. When FALSE Output Motor I-Flag Cluster contains the current PMAC definitions.

Input Motor I-Flag Cluster Input PmacMotorIVarFlag cluster for I-Variable Ix25 motor encoder flags

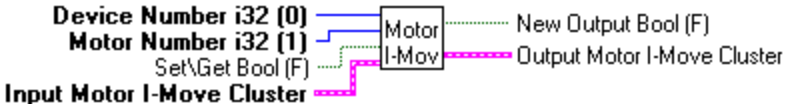
current state

Output Motor I-Flag Cluster Output PmacMotorIVarFlag cluster for I-Variable Ix25 motor encoder flags

New Output Bool (F) TRUE when Output Motor I-Flag Cluster contains new data.

PmacMotorIVarMove

Follow PMACPanel's standard I-Variable VI architecture. When Set/Get is TRUE the movement I-Variables for the specified Motor Number are set. Otherwise they are fetched from PMAC and provided by Output Motor I-Move Cluster with New Output TRUE. Set/Get is not required and defaults to a Get operation.



Motor Number i32 (1) Motor number whose I-Variables are addressed

Device Number i32 (0) Provided by PmacDevOpen

Input Motor I-Move Cluster Input PmacMotorIVarMove cluster for motor move I-Variables

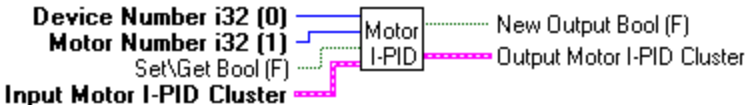
Set\Get Bool (F) When TRUE PMAC is set using Input Motor I-Move Cluster. When FALSE Output Motor I-Move Cluster contains the current PMAC definitions.

New Output Bool (F) TRUE when Output Motor I-Move Cluster contains new data.

Output Motor I-Move Cluster Output PmacMotorIVarMove cluster for motor move I-Variables

PmacMotorIVarPID

Follow PMACPanel's standard I-Variable VI architecture. When Set/Get is TRUE the PID I-Variables for the specified Motor Number are set. Otherwise they are fetched from PMAC and provided by Output Motor I-PID Cluster with New Output TRUE. Set/Get is not required and defaults to a Get operation.



Motor Number i32 (1) Motor number whose I-Variables are addressed

Device Number i32 (0) Provided by PmacDevOpen

Input Motor I-PID Cluster Input PmacMotorIVarPID cluster for motor PID I-Variables

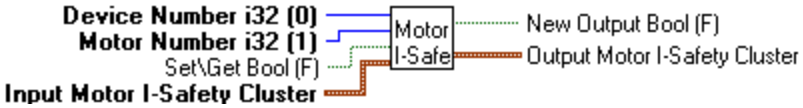
Set\Get Bool (F) When TRUE PMAC is set using Input Motor I-PID Cluster. When FALSE Output Motor I-PID Cluster contains the current PMAC definitions.

New Output Bool (F) TRUE when Output Motor I-PID Cluster contains new data.

Output Motor I-PID Cluster Output PmacMotorIVarPID cluster for motor PID I-Variables

PmacMotorIVarSafety

Follow PMACPanel's standard I-Variable VI architecture. When Set/Get is TRUE the safety I-Variables for the specified Motor Number are set. Otherwise they are fetched from PMAC and provided by Output Motor I-Safety Cluster with New Output TRUE. Set/Get is not required and defaults to a Get operation.



Motor Number i32 (1) Motor number whose I-Variables are addressed

Device Number i32 (0) Provided by PmacDevOpen

Input Motor I-Safety Cluster Input PmacMotorIVarSafety cluster for motor safety I-Variables

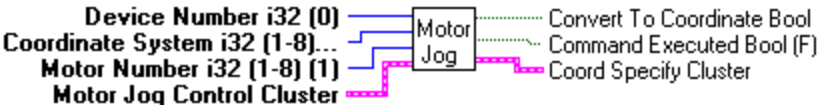
Set\Get Bool (F) When TRUE PMAC is set using Input Motor I-Safety Cluster. When FALSE Output Motor I-Safety Cluster contains the current PMAC definitions.

New Output Bool (F) TRUE when Output Motor I-Safety Cluster contains new data.

Output Motor I-Safety Cluster Output PmacMotorIVarSafety cluster for motor safety I-Variables

PmacMotorJogControl

Generate PMAC on-line commands for controlling jogging Motor Number. Command Executed Bool is TRUE when any button is clicked in Motor Jog Cluster. The value in the numeric control specifies the position Jog To and Jog Relative jog the motor to. This value is interpreted as either Encoder Counts (Default) or Coordinate Units in Coord Number as specified on the button. The button state is provided as the output Convert To Coordinate. This VI builds a Coord Specify Cluster using the various inputs to simplify the interface to PmacMotorPVE and other PMACPanel ICVs.



Device Number i32 (0) Provided by PmacDevOpen

Motor Number i32 (1-8) (1) Motor number to jog

Motor Jog Control Cluster PmacMotorJogControl Cluster of latched buttons to control motor jogging

Coordinate System i32 (1-8) (1) Coordinate System number to interpret jog positions in

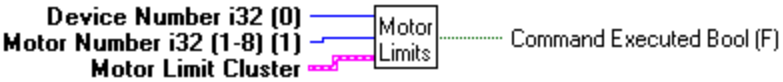
Convert To Coordinate Bool Copy of the Encoder Counts button to indicate whether motor should be represented in CS units.

Command Executed Bool (F) TRUE when any Motor Jog Cluster button is clicked

Coord Specify Cluster Cluster defining the motor, CS, and conversion state applied

PmacMotorLimitControl

Generate PMAC on-line commands for controlling move limits and operation for Motor Number. Command Executed Bool is TRUE when any button is clicked in Motor Limit Cluster.



Device Number i32 (0) Provided by PmacDevOpen

Motor Number i32 (1-8) (1) Motor number to control

Motor Limit Cluster PmacMotorLimitControl Cluster of

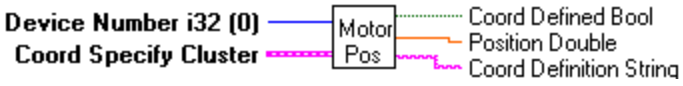
latched buttons to control motor homing and activation state

Command Executed Bool (F) TRUE when any Motor Limit Cluster button is clicked

PmacMotorPosition

Query PMAC for Motor Number's position. PMAC reports the value of the actual position register plus the position bias register plus the compensation correction register, and if bit 16 of Ix05 is 1 (handwheel offset mode), minus the master position register.

Coord Specify Cluster specifies a motor within a CS and an attempt to convert motor position from encoder counts to CS units. If the motor is not defined in the CS no conversion is applied. If the motor is defined and Convert is TRUE Coord Defined is TRUE and position is scaled from encoder counts to CS units. Coord Definition is a string specifying position units as "Encoder" or the CS definition of the motor.



Device Number i32 (0) Provided by PmacDevOpen

Coord Specify Cluster Cluster defining the motor, CS, and conversion state to be applied

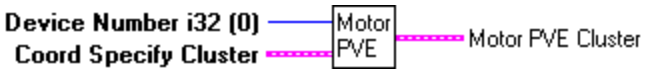
Coord Defined Bool If Motor Number is defined in Coord Number and Convert Bool is TRUE this is TRUE.

Coord Definition String String defining the motor within the CS. "Encoder" if not defined.

Position Double This value is the actual position register plus the position bias register plus the compensation correction register, and if bit 16 of Ix05 is 1 (handwheel offset mode), minus the master position register. The value is in CS units if Convert is TRUE and the motor is defined in the CS. Otherwise the value is in encoder counts.

PmacMotorPVE

Query PMAC for the position, velocity, and following error for Motor Number operating in Coord Number. Assemble the measurements into Motor PVE Cluster. If Convert is TRUE convert the measurements to CS units. Otherwise leave them in encoder counts. See the documentation for PmacMotorPosition, PmacMotorVelocity, and PmacMotorError for details on how these individual values are produced.



Device Number i32 (0) Provided by PmacDevOpen

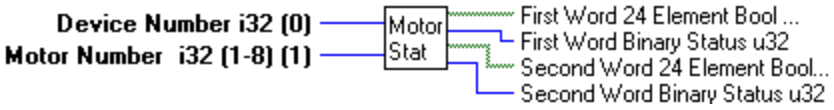
Coord Specify Cluster Cluster defining the motor, CS, and

conversion state to be applied

Motor PVE Cluster Cluster for PmacMotorPVE.ctl

PmacMotorStat

Query PMAC for the status of Motor Number. Report the two status words as arrays of Booleans and unsigned 32 bit integers.


Device Number i32 (0)
Motor Number i32 (1-8) (1)
Motor Stat
First Word 24 Element Bool ...
First Word Binary Status u32
Second Word 24 Element Bool...
Second Word Binary Status u32

Device Number i32 (0) Provided by PmacDevOpen

Motor Number i32 (1-8) (1) Motor to obtain status for

First Word 24 Element Bool Array 24 element Boolean array containing first status word

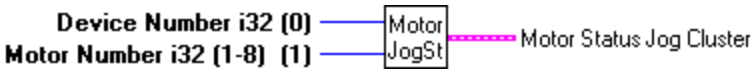
Second Word 24 Element Bool Array 24 element Boolean array containing second status word

First Word Binary Status u32 integer representation of first word status

Second Word Binary Status u32 integer representation of second word status

PmacMotorStatJog

Create a status indicator cluster for the PmacMotorStatJog indicator containing the status for Motor Number.



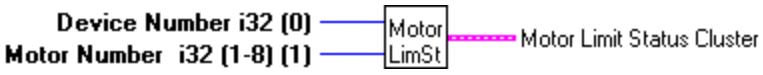
Device Number i32 (0) Provided by PmacDevOpen

Motor Number i32 (1-8) (1) Motor to obtain status for

Motor Status Jog Cluster Cluster for PmacMotorStatJog

PmacMotorStatLimit

Create a status indicator cluster for the PmacMotorStatLimit indicator containing the status for Motor Number.



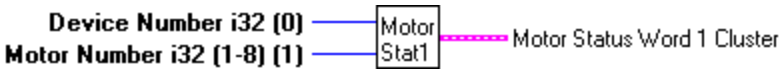
Device Number i32 (0) Provided by PmacDevOpen

Motor Number i32 (1-8) (1) Motor to obtain status for

Motor Limit Status Cluster Cluster for PmacMotorStatLimit

PmacMotorStatWord1

Create a status indicator cluster for the PmacMotorStat1 indicator containing the status for Motor Number.



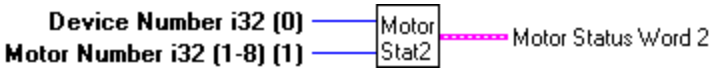
Device Number i32 (0) Provided by PmacDevOpen

Motor Number i32 (1-8) (1) Motor to obtain status for

Motor Status Word 1 Cluster Cluster for PmacMotorStat1

PmacMotorStatWord2

Create a status indicator cluster for the PmacMotorStat2 indicator containing the status for Motor Number.



Device Number i32 (0) Provided by PmacDevOpen

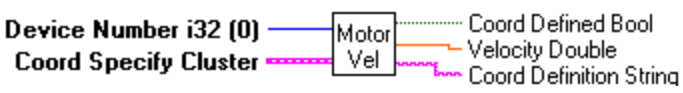
Motor Number i32 (1-8) (1) Motor to obtain status for

Motor Status Word 2 Cluster for PmacMotorStat2

PmacMotorVelocity

Query PMAC for Motor Number's present actual motor velocity, scaled in counts/servo cycle, rounded to the nearest tenth. The raw response reports the contents of the motor actual velocity register (divided by [Ix09*32]). This is converted to counts/msec by multiplying by 8,388,608 and dividing by the I10 default 3,713,707. If I10 is changed modify this value in the diagram.

Coord Specify Cluster specifies a motor within a CS and an attempt to convert motor velocity from encoder counts/mS to CS units/mS. If the motor is not defined in the CS no conversion is applied. If the motor is defined and Convert is TRUE Coord Defined is TRUE and velocity is scaled from encoder counts/mS to CS units/mS. Coord Definition is a string specifying velocity units as "Encoder" or the CS definition of the motor.



Device Number i32 (0) Provided by PmacDevOpen

Coord Specify Cluster Cluster defining the motor, CS, and conversion state to be applied

Coord Defined Bool If Motor Number is defined in Coord Number and Convert Bool is TRUE this is TRUE.

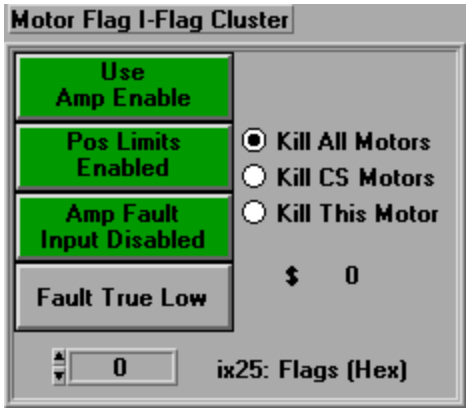
Coord Definition String String defining the motor within the CS. "Encoder" if not defined.

Velocity Double This value is the Present Actual Velocity scaled to encoder counts/mS. Motor velocity is in CS units/mS if Convert is TRUE and the motor is defined in the CS. Otherwise motor velocity in encoder counts/mS.

Indicator and Control Clusters

PmacMotorIVarFlag

This cluster contains items for displaying and modifying the ix25 - motor encoder flag I-Variables. This parameter tells PMAC what set of inputs it will look to for motor X's over-travel limit switches, home flag, amplifier-fault flag, and amplifier-enable output. Typically, these are the flags associated with an encoder input; specifically, those of the position feedback encoder for the motor.



Motor Flag I-Flag Cluster This cluster contains items for displaying and modifying the ix25 - motor encoder flag I-Variables. This parameter tells PMAC what set of inputs it will look to for motor X's over-travel limit switches, home flag, amplifier-fault flag, and amplifier-enable output. Typically, these are the flags associated with an encoder input; specifically, those of the position feedback encoder for the motor.

Use Amplifier Enable Amplifier Enable Use Bit: With bit 16 equal to zero -- the normal case -- the AENAn/DIRn output is used as an amplifier-enable line: off when the motor is "killed", on when it is enabled. Voltage polarity is determined by jumper(s)

E17.
If bit 16 (value $10000, or 65536) is set to one (e.g. I125=$1C000), this output is not used as an amplifier-enable line. This permits use of the line as a direction bit for applications requiring magnitude- and direction outputs, such as driving steppers through
voltage-to-frequency converters. (Setting bit 16 of Ix02 to 1 enables use of this output as a direction bit.) General-purpose use of this output is also possible by assigning an M-variable to it.

Enable Position Limits Over-travel Limit Use Bit: With bit 17 equal to zero -- the normal case -- the +/- LIMn inputs must be held low to permit commanded motion in the appropriate direction. If there are not actual (normally closed or normally conducting) limit switches, the inputs must be hardwired to ground. The direction sense of the limit inputs is the opposite of what many people consider intuitive. That is, the +LIMn input, when taken high (opened), stops commanded motion in the negative direction; the -LIMn input, when taken high, stops commanded motion in the positive direction. It is important to confirm the direction sense of your limit inputs in actual operation.
If bit 17 (value $20000, or 131072) is set to one (e.g. I125=$2C000), motor x does not use these inputs as over-travel limits. This can be done temporarily, as when using a limit as a homing flag. If the limit function is not used at all, these inputs can be used as general-purpose inputs by assigning M-variables to them.

Enable Amplifier Fault Amplifier Fault Use Bit: If bit 20 of Ix25 is 0, the amplifier-fault input function through the FAULTn input is enabled. If bit 20 (value $100000, or 1,048,576) is 1 (e.g. I125=$10C000), this function is disabled. General-purpose use of this input is then possible by assigning an M-variable to the input.

Fault Input Level Amplifier-Fault Polarity Bit: Bit 23 (value 8,388,608) of Ix25 controls the polarity of the amplifier fault input. A zero in this bit means a low-true input (low means fault); a one means high-true (high means fault). The input is pulled high internally, so if no line is attached to the input, and bit 20 of Ix25 is zero (enabling the fault function), bit 23 of Ix25 must be zero to permit operation of the motor.

On Fault Action-on-Fault Bits: Bits 21 (value $200000, or 2,097,152) and 22 (value $400000, or 4,194,344) of Ix25 control what action is taken on an amplifier fault for the motor, or on exceeding the

fatal following error limit (Ix11) for the motor:

Bit 22	Bit 21	Functions
Bit 22=0	Bit 21=0:	Kill all PMAC motors
Bit 22=0	Bit 21=1:	Kill all motors in same CS
Bit 22=1	Bit 21=0:	Kill only this motor
Bit 22=1	Bit 21=1:	Kill only this motor

Regardless of the setting of these bits, a program running in the coordinate system of the offending motor will be halted on an amplifier fault or the exceeding of a fatal following error limit.

I32 **Flag Address** Variable Hex Decimal Limit and Flag set

I125	$C000	(49152)	(LIM1, HMFL1...)
I225	$C004	(49156)	(LIM2, HMFL2...)
I325	$C008	(49160)	(LIM3, HMFL3...)
I425	$C00C	(49164)	(LIM4, HMFL4...)
I525	$C010	(49168)	(LIM5, HMFL5...)
I625	$C014	(49172)	(LIM6, HMFL6...)
I725	$C018	(49176)	(LIM7, HMFL7...)
I825	$C01C	(49180)	(LIM8, HMFL8...)

The addresses for the first eight sets of inputs are given in the default table, above. The addresses for input sets 9 through 16, which are on the Accessory 24 Axis Expansion Board, are given below:

Limit and Flag set	Hex	Decimal
LIM9, HMFL9	$C020	(49184)
LIM10, HMFL10	$C024	(49188)
LIM11, HMFL11	$C028	(49192)
LIM12, HMFL12	$C02C	(49196)
LIM13, HMFL13	$C030	(49200)
LIM14, HMFL14	$C034	(49204)
LIM15, HMFL15	$C038	(49208)
LIM16, HMFL16	$C03C	(49212)

The over-travel-limit inputs specified by this parameter must be held low in order for motor x to be able to command movement. The polarity of the amplifier-fault input is determined by a high-order bit of this parameter (see below). The polarity of the home-flag input is determined by the Encoder/Flag I-Variables 2 and 3 for the specified encoder. The polarity of the amplifier-enable output is determined by Jumper E17.

Extended Addressing: The source address of the flag information occupies bits 0 to 15 of Ix25 (range $0000 to $FFFF, or 0 to 65535). If this is all that is specified -- that is, all higher bits are zero -- then all of the flags are used, and used in the "normal" mode (low-true FAULT, disabling all motors). If higher bits are set to one, some of the flags are not used, or used in an alternate manner, as documented below.

I32 **ix25: Flags (Hex)** This is the hex value of Ix25.

This parameter tells PMAC what set of inputs it will look to for motor X's over-travel limit switches, home flag, amplifier-fault flag, and amplifier-enable

output. Typically, these are the flags associated with an encoder input; specifically, those of the position feedback encoder for the motor.

PmacMotorIVarMove

This cluster contains items for displaying and modifying the most common motor movement I-Variables.

Motor Move I-Var Cluster This cluster contains items for displaying and modifying the most common motor movement I-Variables.

ix20: Jog/Home t-Accel (mS) This parameter establishes the time spent in acceleration in a jogging, homing, or programmed RAPID-mode move (starting, stopping, and changing speeds). However, if Ix21 (jog/home S-curve time) is greater than half this parameter, the total time spent in acceleration will be 2 times Ix21. Therefore, if Ix20 is set to 0, Ix21 alone controls the acceleration time in "pure" S-curve form. In addition, if the maximum acceleration rate set by these times exceeds what is permitted for the motor (Ix19), the time will be increased so that Ix19 is not exceeded.
Do not set both Ix20 and Ix21 to 0 simultaneously, even if you are relying on Ix19 to limit your acceleration, or a division-by-zero error will occur in the jog move calculations, possibly resulting in erratic motion.
A change in this parameter will not take effect until the next move command. For instance, if you wanted a different deceleration time from acceleration time in a jog move, you would specify the acceleration time, command the jog, change the deceleration time, then command the jog move again (e.g. J=), or at least the end of the jog (J/).

ix21: Jog/Home t-S Curve (mS) This parameter establishes the time spent in each "half" of the "S" for S-curve acceleration in a jogging, homing, or RAPID-mode move (starting, stopping, and changing

speeds). If this parameter is more than half of Ix20, the total acceleration time will be 2 times Ix21, and the acceleration time will be "pure" S-curve (no constant acceleration portion). If the maximum acceleration rate set by Ix20 and Ix21 exceeds what is permitted for the motor (Ix19), the time will be increased so that Ix19 is not exceeded.
Do not set both Ix20 and Ix21 to 0 simultaneously, even if you are relying on Ix19 to limit your acceleration, or a division-by-zero error will occur in the jog move calculations, possibly resulting in erratic motion.
A change in this parameter will not take effect until the next move command. For instance, if you wanted a different deceleration time from acceleration time in a jog move, you would specify the acceleration time, command the jog, change the deceleration time, then command the jog move again (e.g. J=), or at least the end of the jog (J/)

ix22: Jog Vel (Ct/mS) This parameter establishes the commanded speed of a jog move, or a programmed RAPID-mode move (if I50=0) for motor x.
Direction of the jog move is controlled by the jog command.
A change in this parameter will not take effect until the next move command. For instance, if you wanted to change the jog speed on the fly, you would start the jog move, change this parameter, then issue a new jog command.

ix23: Home Vel/Dir (Ct/mS) This parameter establishes the commanded speed and direction of a homing-search move for motor x. Changing the sign reverses the direction of the homing move -- a negative value specifies a home search in the negative direction; a positive value specifies the positive direction.

ix26: Home Offset (1/16 Ct) This is the relative position of the end of the homing cycle to the position at which the home trigger was made. That is, the motor will command a stop at this distance from where it found the home flag(s), and call this commanded location as motor position zero.
This permits the motor zero position to be at a different location from the home trigger position, particularly useful when using an over-travel limit as a home flag (offsetting out of the limit before re-enabling the limit input as a limit). If large enough (greater than 1/2 times home speed times accel time) it permits a homing move without any reversal of direction.

ix28: In-POs Band (1/16 Ct) This is the magnitude of the maximum following error at which motor x

will be considered "in position" when not performing a move. Several things happen when the motor is "in-position". First, a status bit in the motor status word is set. Second, if all other motors in the same coordinate system are also "in-position", a status bit in the coordinate system status word is set. Third, for the hardware-selected (FPD0/-FPD3/) coordinate system -- if I2=0 -- or for the software addressed (&n) coordinate system -- if I2=1 -- outputs to the control panel port and to the interrupt controller are set.

Technically, four conditions must be met for a motor to be considered "in-position":

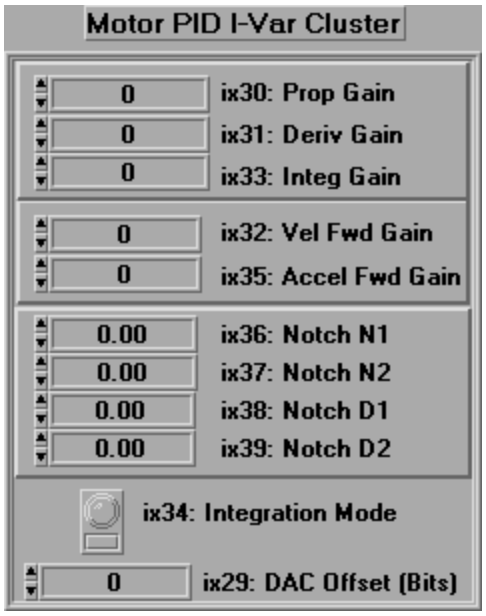
1. the motor must be in closed-loop control
2. the desired velocity must be zero;
3. the magnitude of the following error must be less than this parameter;
4. the move timer must not be active.

The move timer is active during any programmed or non-programmed move, including DWELLs and DELAYs in a program -- if you wish this bit to come true during a program, you must do an indefinite wait between some moves by keeping the program trapped in a WHILE loop that has no moves or DWELLs. More sophisticated in-position functions (for instance, ones that require several consecutive scans within the band) can be implemented using PLC programs. See the program examples section.

The units of this parameter are 1/16 of a count, so the value should be 16 times the number of counts in the in-position band.

PmacMotorIVarPID

This cluster contains items for displaying and modifying the most common motor PID I-Variables.



Motor PID I-Var Cluster This cluster contains items for displaying and modifying the most common motor PID I-Variables.

ix30: Prop Gain (219/Ix08) DAC bits/Encoder count
This term provides a control output proportional to the position error (commanded position minus actual position) of motor x. It acts effectively as an electronic spring. The higher Ix30 is, the stiffer the "spring" is.
The default value of 2000 for this parameter is exceedingly "weak" for most systems (all but the highest resolution velocity-loop systems), causing sluggish motion and/or following error failure. Most users will immediately want to raise this parameter significantly even before starting serious tuning.
This parameter is usually set initially using the Tuning utility in the PMAC Executive Program. It may be changed on the fly at any time to create types of adaptive control.
If the servo update time is changed, Ix30 will have the same effect for the same numerical value. However, smaller update times (faster update rates) should permit higher values of Ix30 (stiffer systems) without instability problems.

ix31: Deriv Gain 226/(Ix30*Ix09) DAC bits/(Counts/cycle)
This term subtracts an amount from the control output proportional to the measured velocity of motor x. It acts effectively as an electronic damper. The higher Ix31 is, the heavier the damping effect is.
If the motor is driving a properly tuned tachometer-loop (velocity) amplifier, the amplifier will provide sufficient damping, and Ix31 should be set to zero. If

the motor is driving a current-loop (torque) amplifier, the amplifier will provide no damping, and Ix31 must be greater than zero to provide damping for stability. On a typical system with a current-loop amplifier and PMAC's default servo update time (~440 msec), an Ix31 value of 2000 to 3000 will provide a critically damped step response.
If the servo update time is changed, Ix31 must be changed proportionately in the opposite direction to keep the same damping effect. For instance, if the servo update time is cut in half, from 440 msec to 220 msec, Ix31 must be doubled to keep the same effect.
This parameter is usually set initially using the Tuning utility in the PMAC Executive Program. It may be changed on the fly at any time to create types of adaptive control.

ix33: Integ Gain 242/(Ix30*Ix08) DAC bits/(counts*cycles)
This term adds an amount to the control output proportional to the time integral of the position error for motor x. The magnitude of this integrated error is limited by Ix63. With Ix63 at a value of zero, the contribution of the integrator to the output is zero, regardless of the value of Ix33.
No further errors are added to the integrator if the output saturates (if output equals Ix69), and, if Ix34=1, when a move is being commanded (when desired velocity is not zero). In both of these cases, the contribution of the integrator to the output remains constant.
If the servo update time is changed, Ix33 must be changed proportionately in the same direction to keep the same effect. For instance, if the servo update time is cut in half, from 440 msec to 220 msec, Ix33 must be cut in half to keep the same effect.
This parameter is usually set initially using the Tuning utility in the PMAC Executive Program. It may be changed on the fly at any time to create types of adaptive control.

ix32: Vel Fwd Gain 226/(Ix30*Ix08) DAC bits/(counts/cycle)
This term adds an amount to the control output proportional to the desired velocity of motor x. It is intended to reduce tracking error due to the damping introduced by Ix31, analog tachometer feedback, or physical damping effects.
If the motor is driving a current-loop (torque) amplifier, Ix32 will usually be equal to (or slightly greater than) Ix31 to minimize tracking error. If the motor is driving a tachometer-loop (velocity) amplifier, Ix32 will typically be substantially greater than Ix31 to minimize tracking error.
If the servo update time is changed, Ix32 must be

changed proportionately in the opposite direction to keep the same effect. For instance, if the servo update time is cut in half, from 440 msec to 220 msec, Ix32 must be doubled to keep the same effect.
This parameter is usually set initially using the Tuning utility in the PMAC Executive Program. It may be changed on the fly at any time to create types of adaptive control.

ix35: Accel Fwd Gain 226/(Ix30*Ix08) DAC bits/(counts/cycle2)
This term adds an amount to the control output proportional to the desired acceleration for motor x. It is intended to reduce tracking error due to inertial lag.
If the servo update time is changed, Ix35 must be changed by the inverse square to keep the same effect. For instance, if the servo update time is cut in half, from 440 msec to 220 msec, Ix35 must be quadrupled to keep the same effect.
This parameter is usually set initially using the Tuning utility in the PMAC Executive Program. It may be changed on the fly at any time to create types of adaptive control.

ix36: Notch N1 This term, along with Ix37-Ix39, is part of the notch filter for motor x, whose purpose is to damp out a resonant mode in the system. This parameter can be set according to instructions in the Servo Loop Features section of the manual.
The notch filter parameters Ix36-Ix39 are 24-bit variables, with 1 sign bit, 1 integer bit, and 22 fractional bits, providing a range of -2.0 to +2.0.
This parameter is usually set initially using the Tuning utility in the PMAC Executive Program. It may be changed on the fly at any time to create types of adaptive control.

ix37: Notch N2 This term, along with Ix36-Ix39, is part of the notch filter for motor x, whose purpose is to damp out a resonant mode in the system. This parameter can be set according to instructions in the Servo Loop Features section of the manual.
The notch filter parameters Ix36-Ix39 are 24-bit variables, with 1 sign bit, 1 integer bit, and 22 fractional bits, providing a range of -2.0 to +2.0.
This parameter is usually set initially using the Tuning utility in the PMAC Executive Program. It may be changed on the fly at any time to create types of adaptive control.

ix38: Notch D1 This term, along with Ix36-Ix39, is part of the notch filter for motor x, whose purpose is to damp out a resonant mode in the system. This parameter can be set according to instructions in the Servo Loop Features section of the manual.

The notch filter parameters Ix36-Ix39 are 24-bit variables, with 1 sign bit, 1 integer bit, and 22 fractional bits, providing a range of -2.0 to +2.0. This parameter is usually set initially using the Tuning utility in the PMAC Executive Program. It may be changed on the fly at any time to create types of adaptive control.

ix39: Notch D2 This term, along with Ix36-Ix39, is part of the notch filter for motor x, whose purpose is to damp out a resonant mode in the system. This parameter can be set according to instructions in the Servo Loop Features section of the manual.
The notch filter parameters Ix36-Ix39 are 24-bit variables, with 1 sign bit, 1 integer bit, and 22 fractional bits, providing a range of -2.0 to +2.0. This parameter is usually set initially using the Tuning utility in the PMAC Executive Program. It may be changed on the fly at any time to create types of adaptive control.

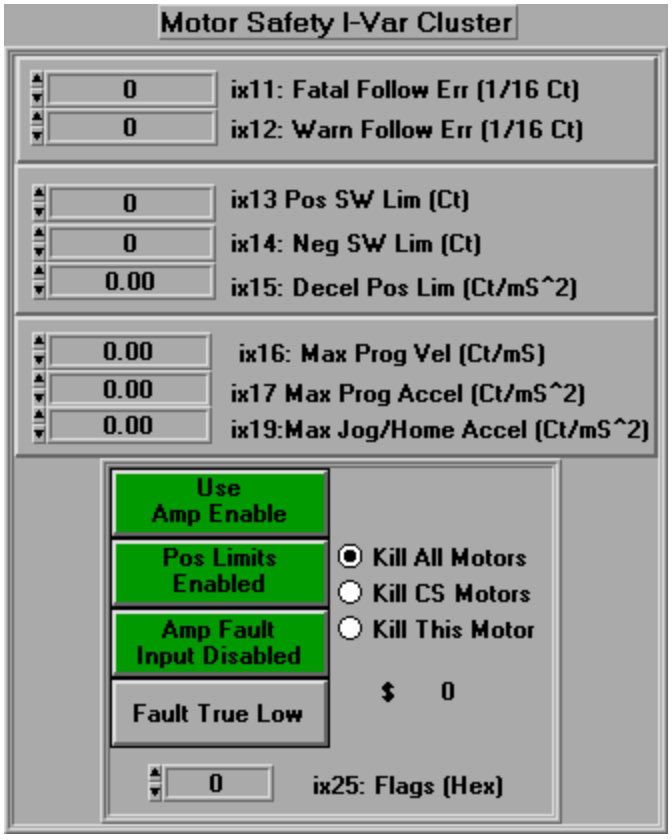
ix34: Integration Mode This parameter controls when the position-error integrator is turned on. If it is 1, position error integration is performed only when PMAC is not commanding a move (when desired velocity is zero). If it is 0, position error integration is performed all the time.
If Ix34 is 1, it is the input to the integrator that is turned off during a commanded move, which means the output control effort of the integrator is kept constant during this period (but is generally not zero). This same action takes place whenever the total control output saturates at the Ix69 value.
This parameter is usually set initially using the Tuning utility in the PMAC Executive Program. When performing the feedforward tuning part of that utility, it is important to set Ix34 to 1 so the dynamic behavior of the system may be observed without integrator action. Ix34 may be changed on the fly at any time to create types of adaptive control.

ix29: DAC Offset (Bits) This parameter is the digital equivalent of an offset potentiometer on the analog output. It can be used to correct for differences in zero-reference between PMAC's analog output and the amplifier's analog input. This offset is active in both closed-loop and open-loop modes, even when the motor is killed.
For a motor not commutated by PMAC (Ix01=0), this is the value that is added onto the output of the servo algorithm or the open loop output value (including the zero output when the motor is killed) before it is sent to the DAC.
If the analog output is unidirectional (bit 16 of Ix02 is 1), this bias term is added before the absolute value function is performed. It is used if there is a

directional bias on the motor. In this type of motor, Ix79 (offset after absolute value) is used to control output deadband or dithering.
For a PMAC-commutated motor (Ix01=1), this is the value that is added onto the first-phase output of the commutation algorithm (Ix79 is added onto the second phase). In addition to the primary use of compensating for analog offsets, it can be used in certain phasing search or phasing direction algorithms for permanent-magnet brushless motors, because it drives the motor like a stepper motor.

PmacMotorIVarSafety

This cluster contains items for displaying and modifying the most common motor safety I-Variables.



Motor Safety I-Var Cluster This cluster contains items for displaying and modifying the most common motor safety I-Variables.

ix11: Fatal Follow Err (1/16 Ct) This parameter sets the magnitude of the following error for motor x at which operation will shut down. When the magnitude of the following error exceeds Ix11, motor x is disabled (killed). If the motor's coordinate system is executing a program at the time, the

program is aborted. It is optional whether other PMAC motors are disabled when this motor exceeds its following error limit; bits 21 and 22 of Ix25 control what happens to the other motor (the default is that all PMAC motors are disabled).
A status bit for the motor, and one for the coordinate system (if the motor is in one) are set. If this coordinate system is hardware-selected on JPAN (with I2=0), or software-addressed by the host (with I2=1), the ERLD/ output on JPAN, and the EROR input to the interrupt controller (except for PMAC-VME) are triggered.
Setting Ix11 to zero disables the fatal following error limit for the motor. This may be desirable during initial development work, but it is strongly discouraged in an actual application. A fatal following error limit is a very important protection against various types of faults, such as loss of feedback, that cannot be detected directly, and that can cause severe damage to people and equipment.
The units of Ix11 are 1/16 of a count. Therefore this parameter must hold a value 16 times larger than the number of counts at which the limit will occur. For example, if the limit is to be 1000 counts, Ix11 should be set to 16,000.

ix12: Warn Follow Err (1/16 Ct) This parameter sets the magnitude of the following error for motor x at which a warning flag goes true. If this limit is exceeded, status bits are set for the motor and the motor's coordinate system (if any). The coordinate system status bit is the logical OR of the status bits of all the motors in the coordinate system.
Setting this parameter to zero disables the warning following error limit function. If this parameter is set greater than the fatal following error limit (Ix11), the warning status bit will never go true, because the fatal limit will disable the motor first.
At any given time, one coordinate system's status bit can be output to several places; which system depends on what coordinate system is hardware-selected on the panel input port if I2=0, or what coordinate system is software-addressed from the host (&n) if I2=1. The outputs that work in this way are F1LD/ (pin 23 on connector J2), F1ER (line IR3 into the programmable interrupt controller (PIC) on PMAC-PC, line IR6 into the PIC on PMAC-STD) and, if E28 connects pins 1 and 2, FEFCO/ (on the JMACH connectors).
The units of Ix12 are 1/16 of a count. Therefore this parameter must hold a value 16 times larger than the number of counts at which the limit will occur. For example, if the limit is to be 1000 counts, Ix11 should be set to 16,000.

ix13 Pos SW Lim (Ct) This parameter sets the

position for motor x which if exceeded in the positive direction causes a deceleration to a stop (controlled by Ix15) and allows no further positive position increments or positive output commands as long as the limit is exceeded. If this value is set to zero, there is no positive software limit (if you want 0 as a limit, use 1). This limit is automatically de-activated during homing search moves.
This limit is referenced to the most recent power-up zero position or homing move zero position. The physical position at which this limit occurs is not affected by axis offset commands (e.g. PSET, X=), although these commands will change the reported position value at which the limit occurs.

ix14: Neg SW Lim (Ct) This parameter sets the position for motor x which if exceeded in the negative direction causes a deceleration to a stop (controlled by Ix15) and allows no further negative position increments or negative output commands as long as the limit is exceeded. If this value is set to zero, there is no negative software limit (if you want 0 as a limit, use -1). This limit is automatically de-activated during homing search moves.
This limit is referenced to the most recent power-up zero position or homing move zero position. The physical position at which this limit occurs is not affected by axis offset commands (e.g. PSET, X=), although these commands will change the reported position value at which the limit occurs.

ix15: Decel Pos Lim (Ct/mS^2) This parameter sets the rate of deceleration that motor x will use if it exceeds a hardware or software limit, or has its motion aborted by command (A or <CONTROL-A>). This value should usually be set to a value near the maximum physical capability of the motor. It is not a good idea to set this value past the capability of the motor, because doing so increases the likelihood of exceeding the following error limit, which stops the braking action, and could allow the axis to coast into a hard stop.
Do not set this parameter to zero, or the motor will continue indefinitely after an abort or limit.
Suppose your motor had 125 encoder lines (500 counts) per millimeter, and you wished it to decelerate at 4000 mm/sec2. You would set Ix15 to 4000 mm/sec2 *500 cts/mm * sec2/1,000,000 msec2 = 2 cts/msec2.

ix16: Max Prog Vel (Ct/mS) This parameter sets a limit to the allowed velocity for LINEAR mode programmed moves for motor x, provided I13 equals zero (no move segmentation). If a blended move command in a motion program requests a higher velocity of this motor, all motors in the coordinate

system are slowed down proportionately so that motor x will not exceed this parameter, yet the path will not be changed. This limit does not affect transition-point, circular, or splined moves. The calculation does not take into account any feedrate override (% value other than 100).
If PMAC's circular interpolation function is used at all, then I13 must be greater than zero, and Ix16 will no be active as a velocity limit.
This parameter also sets the speed of a programmed RAPID mode move for the motor, provided that variable I50 is set to 1 (if I50 is set to 0, jog speed parameter Ix22 is used instead). This happens regardless of the setting of I13.

ix19:Max Jog/Home Accel (Ct/mS^2) This parameter sets a limit to the commanded acceleration magnitude for jog and home mo ves, and for RAPID-mode programmed moves, of motor x. If the acceleration times in force at the time (Ix20 and Ix21) request a higher rate of acceleration, this rate of acceleration will be used instead. The calculation does not take into account any feedrate override (% value other than 100).
Since jogging moves are usually not coordinated between motors, many people prefer to specify jog acceleration by rate, not time. To do this, simply set Ix20 and Ix21 low enough that the Ix19 limit is always used. Do not set both Ix20 and Ix21 to 0; or a Division-by-zero error will result in the move calculations, possibly causing erratic operations. The minimum acceleration time settings that should be used are Ix20=1 and Ix21=0.
The default limit of 0.015625 counts/msec2 is quite low and will probably limit acceleration to a lower value than is desired in most systems; most users will eventually raise this limit This low default was used for safety reasons.
Examples
With Ix20 (accel time) at 100 msec, Ix21 (S-curve time) at 0, and Ix22 (jog speed) at 50 counts/msec, a jog command from stop would request an acceleration of (50 cts/msec) / 100 msec, or 0.5 cts/msec2. If Ix19 were set to 0.25, the acceleration would be done in 200 msec, not 100 msec.
With the same parameters in force, an on-the-fly reversal from positive to negative jog would request an acceleration of (50-(-50) cts/msec) / 100 msec, or 1.0 cts/msec2. The limit would extend this acceleration by a factor of 4, to 400 msec.

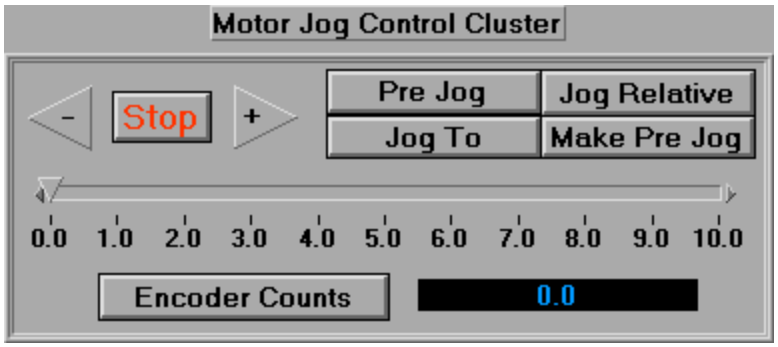
ix17 Max Prog Accel (Ct/mS^2) This parameter sets a limit to the allowed acceleration in LINEAR-mode blended programmed moves for motor x, provided I13 equals zero (no move segmentation). If a LINEAR move command in a motion program

requests a higher acceleration of this motor given its TA and TS time settings, the acceleration for all motors in the coordinate system is stretched out proportionately so that motor x will not exceed this parameter, yet the path will not be changed.
It is possible to have this limit govern the acceleration for all LINEAR-mode moves by setting very low TA and TS times. Do not set both the TA and TS times to zero, or a Division-by-zero error will occur in the move calculations, possibly causing erratic movement. The minimum acceleration time settings that should be used are TA1 with TS0.
The default limit of 0.015625 counts/msec2 is quite low and will probably limit acceleration to a lower value than is desired in most systems; most users will eventually raise this limit This low default was used for safety reasons
When moves are broken into small pieces and blended together, this limit can affect the velocity, because it limits the calculated deceleration for each piece, even if that deceleration is never executed, because it blends into the next piece.
This limit does not affect PVT, CIRCLE, RAPID, or SPLINE moves. The calculation does not take into account any feedrate override (% value other than 100).
If PMAC's circular interpolation function is used at all, then I13 must be greater than zero, and Ix17 will no be active as an acceleration limit.

Motor Flag I-Flag Cluster This cluster contains items for displaying and modifying the ix25 - motor encoder flag I-Variables. This parameter tells PMAC what set of inputs it will look to for motor X's over-travel limit switches, home flag, amplifier-fault flag, and amplifier-enable output. Typically, these are the flags associated with an encoder input; specifically, those of the position feedback encoder for the motor.

PmacMotorJogControl

Cluster of latched buttons to control motor jogging



Motor Jog Control Cluster Cluster of latched buttons to control motor jogging

Jog - The command sent to PMAC is J-.
This command causes the addressed motor to jog in the negative direction indefinitely. Jogging acceleration and velocity are determined by the values of Ix19-Ix22 in force at the time of this command.
PMAC will reject this command if the motor is in a coordinate system that is currently running a motion program (reporting ERR001 if I6 is 1 or 3).

Jog Stop The command sent to PMAC is J/.
This command causes the addressed motor to stop jogging. It also restores position control if the motor's servo loop has been opened (enabled or killed), with the new commanded position set equal to the actual position at the time of the J/ command. Jogging deceleration is determined by the values of Ix19-Ix21 in force at the time of this command.

PMAC will reject this command if the motor is in a coordinate system that is currently running a motion program (reporting ERR001 if I6 is 1 or 3).

Jog + The command sent to PMAC is J+.
This command causes the addressed motor to jog in the positive direction indefinitely. Jogging acceleration and velocity are determined by the values of Ix19-Ix22 in force at the time of this command.
PMAC will reject this command if the motor is in a coordinate system that is currently running a motion program (reporting ERR001 if I6 is 1 or 3).

Pre Jog The command sent to PMAC is J=.
This command causes the addressed motor to jog to the last pre-jog and pre-handwheel-move position (the most recent programmed position). Jogging acceleration and velocity are determined by the values of Ix19-Ix22 in force at the time of this command.
The register containing this position information for the motor is called the target position register (D:$080B for Motor 1, D:$08CB for Motor 2, etc.). Suggested M-variable definitions M163, M263, etc. can be used in programs to give access to these registers. It is possible to write to this register, then issue a J= command, to provide a variable jog-to-position capability. If you do this, make sure that the PMATCH function is executed before the next motion program is run (usually done by setting I14 to 1).
The J= command can also be useful if a program has been aborted in the middle of a move, because it will

move the motor to the programmed move end position, so the program may be resumed properly from that point.
PMAC will reject this command if the motor is in a coordinate system that is currently running a motion program (reporting ERR001 if I6 is 1 or 3).

Jog To The command sent to PMAC is J={position}. This command causes the addressed motor to jog to the position specified by {position}. {position} is determined using the value in the associated numeric control. {position} will be in encoder counts if the neighboring button says Encoder Counts and in CS units if the button says Coordinate Units and the motor is defined in the currently address CS.
Jogging acceleration and velocity are determined by the values of Ix19-Ix22 in force at the time of this command.
PMAC will reject this command if the motor is in a coordinate system that is currently running a motion program (reporting ERR001 if I6 is 1 or 3).

Jog Relative The command sent to PMAC is J:{position}.
This command causes a motor to jog the distance specified by {position} relative to the present commanded position. {position} is determined using the value in the associated numeric control. {position} will be in encoder counts if the neighboring button says Encoder Counts and in CS units if the button says Coordinate Units and the motor is defined in the currently address CS.
Jogging acceleration and velocity are determined by the values of Ix19-Ix22 in force at the time of this command. Compare to J^{position}, which is a jog relative to the present actual position.
PMAC will reject this command if the motor is in a coordinate system that is currently running a motion program (reporting ERR001 if I6 is 1 or 3).

Make Pre Jog The command sent to PMAC is J=={position}.
This command causes the addressed motor to jog to the position specified by {position}. {position} is determined using the value in the associated numeric control. {position} will be in encoder counts if the neighboring button says Encoder Counts and in CS units if the button says Coordinate Units and the motor is defined in the currently address CS.
Jogging acceleration and velocity are determined by the values of Ix19-Ix22 in force at the time of this command.
After jogging to {position} this position becomes the pre-jog position for all subsequent Pre Jog (J=) commands.
PMAC will reject this command if the motor is in a

coordinate system that is currently running a motion program (reporting ERR001 if I6 is 1 or 3).

Jog Position Specify a Jog To, Jog Relative, or Make Pre Jog position. When the neighboring button says Encoder Counts this value is interpreted as encoder counts. If the button says Coordinate Units this value will be interpreted as a position in CS Units if the motor is defined in the currently addressed CS.

Convert To Coordinate When this button says Encoder Counts interpret the value of the numeric control in CS units when executing Jog To, Jog Relative, and Make Pre Jog commands. When the button says Coordinate Units interpret the value as CS Units if the motor is in the currently addressed CS.

PmacMotorLimitControl

Cluster of latched buttons to control motor homing and activation state

Motor Limit Control Cluster Cluster of latched buttons to control motor homing and activation state

Home The command sent to PMAC is HOME. This command causes the addressed motor to perform a homing search routine. The characteristics of the homing search move are controlled by motor I-variables Ix03 and Ix19-Ix26, plus encoder I-variables 2 and 3 for that motor's position encoder. The on-line home command simply starts the homing search routine. PMAC provides no automatic indication that the search has completed (although the In-Position interrupt could be used for this purpose) or whether the move completed successfully. Polling, or a combination of polling and interrupts, is generally used to determine completion and success.
By contrast, when a homing search move is given in a motion program (e.g. HOME1,2), the motion program will keep track of completion by itself as part of its sequencing algorithms.
If there is an axis offset in the axis-definition statement for the motor, the reported position at the end of the homing search move will be equal to the axis offset, not to zero.

Home Zero The command sent to PMAC is HOMEZ.

This command causes the addressed motor to perform a zero-move homing search. Instead of jogging until it finds a pre-defined trigger, and calling its position at the trigger the home position, with this command, the motor calls wherever it is (actual position) at the time of the command the home position.
If there is following error due to offsets when this command is issued, there can be movement of the magnitude of the following error, because it takes the actual position at the time of the command and makes it the commanded position. Offsets can then push the actual position away. A stable system with integral gain will not have this problem.
If there is an axis offset in the axis-definition statement for the motor, the reported position at the end of the homing search move will be equal to the axis offset, not to zero.

Reset The command sent to PMAC is $.
This command causes PMAC to initialize the addressed motor, performing any required commutation phasing and full reading of an absolute position sensor, leaving the motor in a closed-loop zero-velocity state. (For a non-commutated motor with an incremental encoder, the J/ command may also be used.)
This command is necessary to initialize a PMAC-commutated motor after power-up/reset if Ix80 for the motor is set to 0. If Ix80 is 1, the initialization will be done automatically during the power-up/reset cycle.
This command will not be accepted if the motor is executing a move.

Kill The command sent to PMAC is K.
This command causes PMAC to "kill" the outputs for the addressed motor. The servo loop is disabled, the DAC outputs are set to zero (Ix29 and/or Ix79 offsets are still in effect), and the AENA output for the motor is taken to the disable state (polarity is determined by E17).

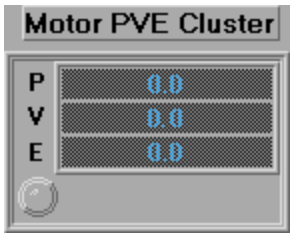
Closed-loop control of this motor can be resumed with a J/ command. The A command will re-establish closed-loop control for all motors in the addressed coordinate system, and the <CTRL-A> command will do so for all motors on PMAC.
The action on a K command is equivalent to what PMAC does automatically to the motor on an amplifier fault or a fatal following error fault.
PMAC will reject this command if the motor is in a coordinate system that is currently running a motion program (reporting ERR001 if I6 is 1 or 3). The program must be stopped first, usually with an A command. However, the global <CTRL-K>

command will kill all motors immediately, regardless of whether any are running motion programs.

PmacMotorPVE

The indicator accepts the cluster provided by the PmacMotorPVE VI. The values are displayed in encoder counts to the nearest tenth or CS units as defined by the coordinate system. The LED indicator is Orange and a text indicator shows the CS definition for the motor if the display is in CS units. Otherwise the indicator is OFF and the text indicator specifies Encoder Counts.

See the documentation for PmacMotorPosition, PmacMotorVelocity, and PmacMotorError for details on how these individual values are produced.



Motor PVE Cluster The indicator cluster displays a motor's position, velocity, and following error. An Orange LED indicates that the displayed values are in CS units. The caption specifies the displayed values as in encoder counts or the actual motor definition in the CS.
See the documentation for PmacMotorPosition, PmacMotorVelocity, and PmacMotorError for details on how these individual values are produced.

Motor Positions Array of numerics for displaying Position, Velocity, and Following Error.
See the documentation for PmacMotorPosition, PmacMotorVelocity, and PmacMotorError for details on how these individual values are produced.

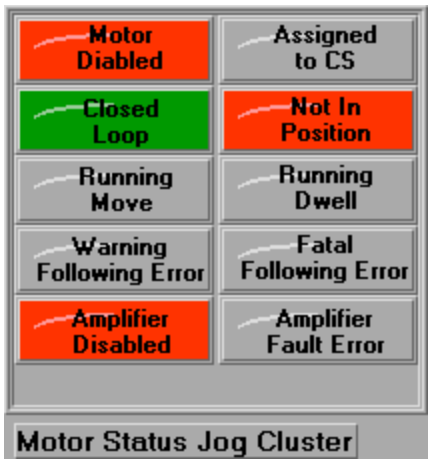
Coord Defined When this LED is Orange there values in the indicators are displayed in CS units.

Unit Label Text item specifying whether the position, velocity, and following error for this motor are in encoder counts or CS units. If they are in CS units the item displays the motor definition.

PmacMotorStatJog

This is an indicator cluster for the most common motor movement and jog status bits. It contains a label to display which motor the status is for.

The colors of the indicators are indicative of the information conveyed by the particular bit. For example 'Motor Disabled' is Red when TRUE and grey otherwise.



Motor Status Jog Cluster This is an indicator cluster for the most common motor jog status bits

Motor Activated Status Bit 23 Word 1 - Motor Activated: Green when Ix00 is 1 and the motor calculations are active; Red when Ix00 is 0 and motor calculations are deactivated.

Assigned to CS Status Bit 23 Word 2 - Assigned to C.S.: Green when the motor has been assigned to an axis in any coordinate system through an axis definition statement. Grey when the motor is not assigned to an axis in any coordinate system.

Open Loop Status Bit 18 Word 1 - Open Loop Mode: Red when the servo loop for the motor is open, either with outputs enabled or disabled (killed). (Refer to Amplifier Enabled status bit to distinguish between the two cases.) Green when the servo loop is closed (under position control, always with outputs enabled).

In Position Status Bit 0 Word 2 - In Position: Green when three conditions are satisfied: the desired velocity zero bit is 1 (which requires closed-loop control and no commanded move); the program timer is off (not currently executing any move, DWELL, or DELAY), and the magnitude of the following error is smaller than Ix28. Red otherwise.

Running Move Status Bit 17 Word 1 - Running Definite-Time Move: Green when the motor is executing any move with a predefined end-point and end-time. This includes any motion program move dwell or delay, any jog-to-position move, and the portion of a homing search move after the trigger has been found. Grey otherwise. It changes from Red to Grey when execution of the commanded move finishes.

Dwell In Progress Status Bit 15 Word 1 - Dwell in Progress: Green when the motor's coordinate system is executing a DWELL instruction. Grey otherwise.

Warning Following Error Status Bit 1 Word 2 - Warning Following Error: Red if the following error for the motor exceeds its warning following error limit (Ix12). It stays at Red if the motor is killed due to fatal following error. Grey at all other times, changing from Red to Grey when the motor's following error reduces to under the limit, or if killed, is re-enabled.

Fatal Following Error Status Bit 2 Word 2 - Fatal Following Error: Red if this motor has been disabled because it exceeded its fatal following error limit (Ix11). Grey at all other times, becoming Grey again when the motor is re-enabled.

Amplifier Enabled Status Bit 14 Word 2 - Amplifier Enabled: Green when the outputs for this motor's amplifier are enabled, either in open-loop or closed-loop mode (refer to Open-Loop Mode status bit to distinguish between the two cases). Red when the outputs are disabled (killed).

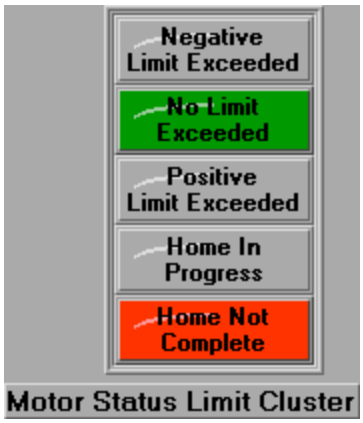
Amp Fault Status Bit 3 Word 2 - Amplifier Fault Error: Red if this motor has been disabled because of an amplifier fault error, even if the amplifier fault signal has gone away. Grey at all other times, becoming Grey again when the motor is re-enabled.

C.S. Text description of the motor number this status cluster is displaying.

PmacMotorStatLimit

This is an indicator cluster for motor limit and homing status bits.

The colors of the indicator LEDs are indicative of the information conveyed by the particular bit. For example 'Negative Limit Exceeded' is Red when TRUE and gray otherwise. Whereas, 'Home Not Complete' is Red when FALSE and Green when TRUE. Furthermore, the text displayed when TRUE is 'Home Complete'.



Motor Status Limit Cluster This is an indicator cluster for the five motor status bits related to limits and homing

Neg Limit Exceeded Status Bit 22 Word 1- Negative End Limit Set: Red when motor actual position is less than the software negative position limit (Ix14), or when the hardware limit on this end (+LIMn -- note!) has been tripped; Grey otherwise.

Stopped on Limit Status Bit 11 Word 2 - Stopped on Position Limit: Red if this motor has stopped because of either a software or a hardware position (over-travel) limit, even if the condition that caused the stop has gone away. Green at all other times, even when into a limit but moving out of it.

Pos Limit Exceeded Status Bit 21 Word 1- Positive End Limit Set: Red when motor actual position is greater than the software positive position limit (Ix13), or when the hardware limit on this end (-LIMn -- note!) has been tripped; Grey otherwise.

Home In Progress Status Bit 10 Word 1- Home Search in Progress: Green while the motor is searching for its home trigger -- it becomes Green as soon as the calculations for the move have started, and becomes Grey again as soon as it has found the trigger (which is before it has finished the entire move). This is not a good bit to observe to see if the homing move is complete. Use the Home Complete bit instead.

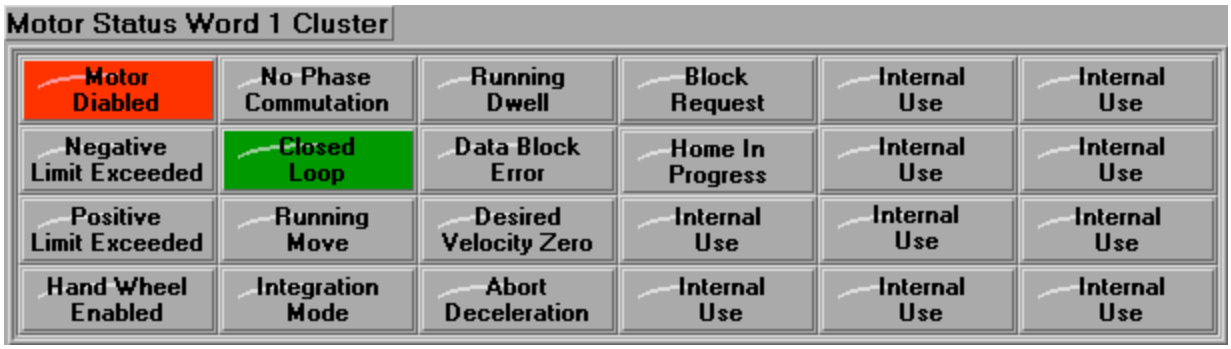
Home Complete Status Bit 10 Word 2 - Home Complete: Red on power-up or reset, becomes Green when a homing search move is successfully completed. If a second homing move is done, this bit is set to Red at the beginning of the move, and only becomes Green again if that homing search move is successfully completed.

PmacMotorStatWord1

This is an indicator cluster for the first status word of a motor

The colors of the indicators are indicative of the information conveyed by the particular bit. For example 'Negative Limit Exceeded' is Red when TRUE and grey otherwise. 'Motor Disabled' is Red when FALSE and Green when TRUE. Furthermore, the text displayed when TRUE is 'Motor Enabled'.

The layout of the indicators represents the location of the bit in the status word. Missing indicators represent status bits that are not defined or for internal use.



Motor Status Word 1 Cluster This is an indicator cluster for the first status word of a motor

Internal Use Bit 0 (Internal use)

Internal Use Bit 1 (Internal use)

Internal Use Bit 2 (Internal use)

Internal Use Bit 3 (Internal use)

Internal Use Bit 4 (Internal use)

Internal Use Bit 5 (Internal use)

Internal Use Bit 6 (Internal use)

Internal Use Bit 7 (Internal use)

Internal Use Bit 8 (Internal use)

Internal Use Bit 9 (Internal use)

Home In Progress Status Bit 10 Word 1 - Home Search in Progress: This bit is 1 while the mo tor is searching for its home trigger -- it becomes 1 as soon as the calculations for the move have started, and becomes 0 again as soon as it has found the trigger (which is before it has finished the entire move). This is not a good bit to observe to see if the homing move is complete. Use the Home Complete bit instead.

Block Request Status Bit 11 Word 1 - Block Request: This bit is 1 when the motor has just entered a new move section, and is requesting that the upcoming section be calculated. It is 0 otherwise. It is primarily for internal use.

Abort Deceleration Status Bit 12 Word 1 - Abort Deceleration: This bit is 1 if the motor is decelerating due to an Abort command, or due to hitting hardware or software position (over-travel) limits. It is 0 otherwise. It changes from 1 to 0 when the commanded deceleration to zero velocity finishes.

Desired Velocity Zero Status Bit 13 Word 1 - Desired Velocity Zero: This bit is 1 if the motor is in closed-loop control and the commanded velocity is zero (i.e. it is trying to hold position). It is zero either if the motor is in closed-loop mode with non-zero commanded velocity, or if it is in open-loop mode.

Data Block Error Status Bit 14 Word 1 - Data Block Error: This bit is 1 when move execution has been aborted because the data for the next move section was not ready in time. This is due to insufficient calculation time. It is 0 otherwise. It changes from 1 to 0 when another move sequence is started.

Dwell In Progress Status Bit 15 Word 1 - Dwell in Progress: This bit is 1 when the motor's coordinate system is executing a DWELL instruction. It is 0 otherwise.

Integration Mode Status Bit 16 Word 1 - Integration Mode: This bit is 1 when Ix34 is 1 and the servo loop integrator is only active when desired velocity is zero. It is 0 when Ix34 is 0 and the servo loop integrator is always active.

Running Move Status Bit 17 Word 1 - Running Definite-Time Move: This bit is 1 when the motor is executing any move with a predefined end-point and end-time. This includes any motion program move dwell or delay, any jog-to-position move, and the portion of a homing search move after the trigger has been found. It is 0 otherwise. It changes from 1 to 0 when execution of the commanded move finishes.

Open Loop Status Bit 18 Word 1 - Open Loop Mode: This bit is 1 when the servo loop for the motor is open, either with outputs enabled or disabled (killed). (Refer to Amplifier Enabled status bit to distinguish between the two cases.) It is 0 when the servo loop is closed (under position control, always with outputs enabled).

Phased Motor Status Bit 19 Word 1 - Phased Motor: This bit is 1 when Ix01 is 1 and this motor is being commutated by PMAC; it is 0 when Ix01 is 0 and this motor is not being commutated by PMAC.

Hand Wheel Enabled Status Bit 20 Word 1 - Handwheel Enabled: This bit is 1 when Ix06 is 1 and position following for this axis is enabled; it is 0 when Ix06 is 0 and position following is disabled.

Pos Limit Exceeded Status Bit 21 Word 1 - Positive End Limit Set: This bit is 1 when motor actual position is greater than the software positive position limit (Ix13), or when the hardware limit on this end (-LIMn -- note!) has been tripped; it is 0 otherwise.

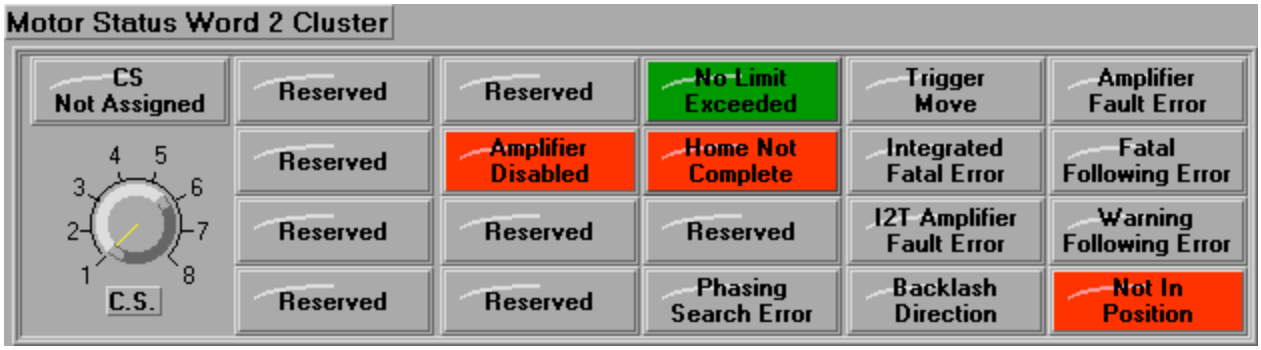
Neg Limit Exceeded Status Bit 22 Word 1 - Negative End Limit Set: This bit is 1 when motor actual position is less than the software negative position limit (Ix14), or when the hardware limit on this end (+LIMn -- note!) has been tripped; it is 0 otherwise.

Motor Activated Status Bit 23 Word 1 - Motor Activated: This bit is 1 when Ix00 is 1 and the motor calculations are active; it is 0 when Ix00 is 0 and motor calculations are deactivated.

PmacMotorStatWord2

This is an indicator cluster for the second status word of a motor.

The colors of the indicators are indicative of the information conveyed by the particular bit. For example 'Negative Limit Exceeded' is Red when TRUE and grey otherwise. 'Not In Position' is Red when FALSE and Green when TRUE. Furthermore, the text displayed when TRUE is 'In Position'.



Motor Status Word 2 Cluster This is an indicator cluster for the second status word of a motor

In Position Status Bit 0 Word 2 - In Position: This bit is 1 when three conditions are satisfied: the

desired velocity zero bit is 1 (which requires closed-loop control and no commanded move); the program timer is off (not currently executing any move, DWELL, or DELAY), and the magnitude of the following error is smaller than Ix28.

Fatal Following Error Status Bit 1 Word 2 - Warning Following Error: This bit is 1 if the following error for the motor exceeds its warning following error limit (Ix12). It stays at 1 if the motor is killed due to fatal following error. It is 0 at all other times, changing from 1 to 0 when the motor's following error reduces to under the limit, or if killed, is re-enabled.

Warning Following Error Status Bit 2 Word 2 - Fatal Following Error: This bit is 1 if this motor has been disabled because it exceeded its fatal following error limit (Ix11). It is 0 at all other times, becoming 0 again when the motor is re-enabled.

Amplifier Fault Status bit 3 Word 2 - Amplifier Fault Error: This bit is 1 if this motor has been disabled because of an amplifier fault error, even if the amplifier fault signal has gone away. It is 0 at all other times, becoming 0 again when the motor is re-enabled.

Backlash Direction

I2T Amplifier Fault Error

Integrated Fault Error

Trigger Move

Phasing Search Error

Reserved Bit 9 (Reserved for future use)

Home Complete Status bit 10 Word 2 - Home Complete: This bit, set to 0 on power-up or reset, becomes 1 when a homing search move is successfully completed. If a second homing move is done, this bit is set to 0 at the beginning of the move, and only becomes 1 again if that homing search move is successfully completed.

Stopped on Limit Status Bit 11 Word 2 - Stopped on Position Limit: This bit is 1 if this motor has stopped because of either a software or a hardware position (over-travel) limit, even if the condition that caused the stop has gone away. It is 0 at all other times, even when into a limit but moving out of it.

Reserved Bit 12 (Reserved for future use)

Reserved Bit 13 (Reserved for future use)

Motor Activated Status Bit 14 Word 2 - Amplifier Enabled: This bit is 1 when the outputs for this motor's amplifier are enabled, either in open-loop or closed-loop mode (refer to Open-Loop Mode status bit to distinguish between the two cases). It is 0 when the outputs are disabled (killed).

Reserved Bit 15 (Reserved for future use)

Reserved Bit 16 (Reserved for future use)

Reserved Bit 17 (Reserved for future use)

Reserved Bit 18 (Reserved for future use)

Reserved Bit 19 (Reserved for future use)

C.S. Status Bits 20 - 22 Word 2 - (C.S. - 1) Number: These three bits together hold a value equal to the (Coordinate System number minus one) to which the motor is assigned. Bit 22 is the MSB, and bit 20 is the LSB. For instance, if the motor is assigned to an axis in C. S. 6, these bits would hold a value of 5: bit 22 =1, bit 21 = 0, and bit 20 = 1.

Coordinate System Assigned Status Bit 23 Word 2 - Assigned to C.S.: This bit is 1 when the motor has been assigned to an axis in any coordinate system through an axis definition statement. It is 0 when the motor is not assigned to an axis in any coordinate system.

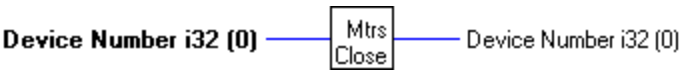
PmacMotors

\PmacMotors - Monitor and plot the motion of collections of motors in defined coordinate systems. Plotting tools for selecting which motors and motion variables to plot are available. Samples of real-time strip charts and XY charts are provided.

VIs

PmacMotorsCloseLoop

Close all PMAC motor loops



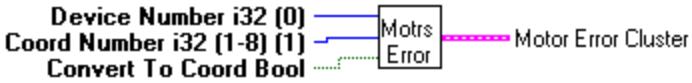
Device Number i32 (0) Provided by PmacDevOpen

Device Number i32 (0) Passed from input Device Number after motor loops are closed

PmacMotorsErrors

Query PMAC for the following errors for all motors. Following error is the difference between a motor's desired and measured position at any instant. When a motor is open-loop (killed or enabled), following error does not exist and PMAC reports a value of 0.

Assemble the measurements into PmacMotorsPVE Cluster. If Convert To Coord is TRUE convert the measurements to CS units for those motors defined in the CS. Otherwise leave them in encoder counts.



Device Number i32 (0) Provided by PmacDevOpen

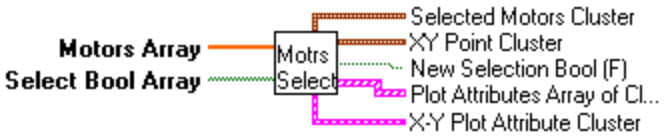
Convert To Coord Bool Convert PMAC responses from encoder counts to CS units for motors in Coord Number

Coord Number i32 (1-8) (1) Coordinate number to use

Motor Error Cluster Cluster for PmacMotorsPVE.ctl to display all motor following errors.

PmacMotorsPlotSelect

Motors Array contains positions, velocities, or following errors for all motors on your PMAC. Select Bool Array defines which motors are copied into Selected Motors Cluster for plotting on LabVIEW strip charts. XY Point Cluster contains two values for X-Y plotting. New Selection is TRUE when Select Bool changes and indicates the Plot Attributes Array of Clusters and X-Y Plot Attribute Cluster contain new information for updating plot legend attributes.



Motors Array Array of motor positions, velocities, or following errors from PmacMotorsPositions, PmacMotorsVelocities, PmacMotorsErrors to be selected.

Select Bool Array Array of Booleans from PmacMotorsSelect specifying which motors to select from Motors Array

Selected Motors Cluster Cluster of doubles containing values for selected motors

XY Point Cluster Cluster containing two selected motors for

X-Y plotting

Plot Attributes Array of Clusters Cluster for setting attributes of Plot based on selected motors.

Plot Attribute Cluster Cluster defining Active Plot, Plot Color, and Plot Name.

Active Plot Active Plot for Plot attribute node

Plot Color Plot Color for Plot attribute node

Plot Name Plot Name for Plot attribute node

New Selection Bool (F) TRUE when Select Bool Array changes. Indicates Plot Attributes Array of Clusters and X-Y Plot Attribute Cluster contain new values for plot legends

X-Y Plot Attribute Cluster Cluster for setting attributes of X-Y Plot based on selected motors.

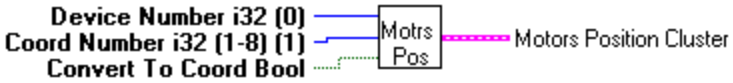
Active Plot Active Plot for Plot attribute node

Plot Name Plot Name for Plot attribute node

PmacMotorsPositions

Query PMAC for the positions for all motors. PMAC reports the value of the actual position register plus the position bias register plus the compensation correction register, and if bit 16 of Ix05 is 1 (handwheel offset mode), minus the master position register.

Assemble the measurements into PmacMotorsPVE Cluster. If Convert To Coord is TRUE convert the measurements to CS units for those motors defined in the CS. Otherwise leave them in encoder counts.



Device Number i32 (0) Provided by PmacDevOpen

Convert To Coord Bool Convert PMAC responses from encoder counts to CS units for motors in Coord Number

Coord Number i32 (1-8) (1) Coordinate number to use

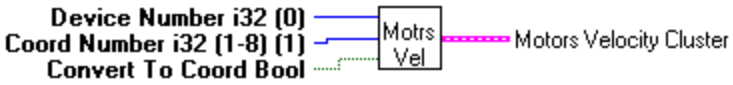
Motors Position Cluster Cluster for PmacMotorsPVE.ctl to display all motor positions.

PmacMotorsVelocities

Query PMAC for all motor's present actual motor velocity, scaled in counts/servo cycle, rounded to the nearest tenth. The raw response reports the contents of the

motor actual velocity register (divided by [Ix09*32]). This is converted to counts/msec by multiplying by 8,388,608 and dividing by the I10 default 3,713,707. If I10 is changed modify this value in the diagram.

Assemble the measurements into PmacMotorsPVE Cluster. If Convert To Coord is TRUE convert the measurements to CS units/mS for those motors defined in the CS. Otherwise leave them in encoder counts/mS.



Device Number i32 (0) Provided by PmacDevOpen

Convert To Coord Bool Convert PMAC responses from encoder counts to CS units for motors in Coord Number

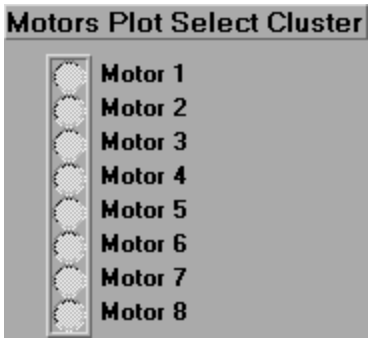
Coord Number i32 (1-8) (1) Coordinate number to use

Motors Velocity Cluster Cluster for PmacMotorsPVE.ctl to display all motor velocities.

Indicator and Control Clusters

PmacMotorsPlotSelect

Array of Boolean controls for selecting motors from PmacMotorsPositions, PmacMotorsVelocities, PmacMotorsErrors.

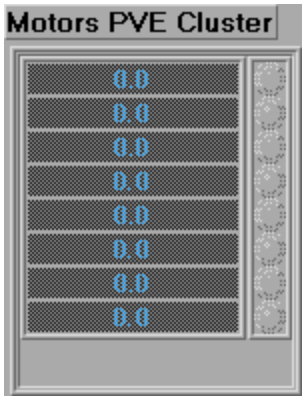


Motors Plot Select Cluster Array of Booleans for selecting motors from PmacMotorsPositions, PmacMotorsVelocities, PmacMotorsErrors.

PmacMotorsPVE

This indicator contains an array of indicators to display the positions, velocities, an/or following errors for all motors. An array of Boolean indicators indicate which motors have been translated from encoder units to CS units. When displaying CS units the caption specifies the currently addressed CS. The arrays can be resized by dragging the corner of the array to display the number of motors in your system.

See the documentation for PmacMotor (s)Position, PmacMotor(s)Velocity, and PmacMotor(s)Error for details on how these individual values are produced.



Motors PVE Cluster The indicator cluster displays an array of values for all PMAC motors. The array may be positions, velocities, or following errors. The array of Boolean indicators indicate which values are in CS units. The caption specifies the displayed values as being in encoder counts or a specific CS.
See the documentation for PmacMotor (s)Position, PmacMotor(s)Velocity, and PmacMotor(s)Error for details on how these individual values are produced.

Motor Value Array Array of numerics for positions, velocities, or following errors for each motor. See the documentation for PmacMotor (s)Position, PmacMotor(s)Velocity, and PmacMotor(s)Error for details on how these individual values are produced.

C.S. Defined Array of Booleans indicating which motors are displayed in CS Units.

C.S. Applied Caption indicating the currently addressed coordinate system or that the displayed values are in Encode Counts.

PmacPLC

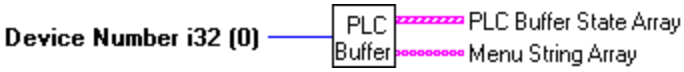
\PmacPLC - PLC programs and their execution status can be edited and controlled using the VIs in this collection.

VIs

PmacPLCBuffer

PMAC maintains a list of PLC program buffers in memory. This organization is a little different from motion program buffer lists. The list identifies the PLC number, start address of the PLC program, and its enable state. This VI builds an array of

PLC Buffer Info clusters for each PLC program loaded in PMAC. The cluster contains Execute State indicating whether the PLC is enabled, PLC Number, and the size of the program in words. Menu String Array contains a string for each loaded PLC and its current enable state.



Device Number i32 (0) Provided by PmacDevOpen

PLC Buffer State Array Array of PLC Buffer Info Clusters for all PLC programs

PLC Buffer Info Cluster containing Size and Run State information about PLC Number

Execute State Bool TRUE indicates that PLC Number is enabled.

PLC Number i32 PLC Number this cluster refers to.

Size i32 Size of PLC program in words

Menu String Array Array of strings for loaded PLCs. Also contains information about whether the PLC is enabled or disabled.

PmacPLCDownLoadBuffer

Write Program Buffer to a temporary file then compile and download to PMAC. Down Load File Path and Down Load Log File Path specify the temporary file's name and compile log. If a compile error occurs a modal pop-up panel containing the log is displayed. PLC VI State Cluster contains identifies PLC Loaded, its PLC Number, and Execute State.

Program Buffer String Program buffer to down load and parse for PLC Number.

DeviceNumber i32 (0) Provided by PmacDevOpen

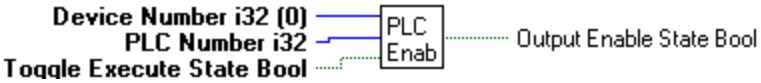
Down Load File Path Path of temporary file used to down load Program Buffer.

Down Load Log File Path Path of file containing down load compile log.

PLC VI State Cluster Cluster defining currently selected PLC and it execution state.

PmacPLCEnable

Check PLC Number for its enable state. When the specified PLC is enabled Output Enable State is TRUE. When Toggle Execute State is TRUE toggle the PLC's enable state from enabled to disabled or disabled to enabled.



PLC Number i32 PLC Number whose execution state is to be toggled.

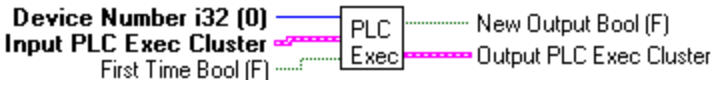
Device Number i32 (0) Provided by PmacDevOpen.

Toggle Execute State Bool When TRUE toggle the execution state of PLC Number. If the PLC was off turn it on. If it was off turn it on.

Output Enable State Bool This flag always reflects the enable state of PLC Number. TRUE indicates that PLC Number is enabled.

PmacPLCExec

This VI controls the execution of foreground and background PLC programs by modifying i5 using a PmacPLCExec control as both an indicator and a control. When First Time is TRUE New Output is TRUE and Output PLC Exec Cluster indicates the state of foreground and background PLC program execution. When either button in Input PLC Exec Cluster doesn't match the last Output PLC Exec Cluster contents the execution state of the foreground or background PLC programs is toggled.



Device Number i32 (0) Provided by PmacDevOpen

Input PLC Exec Cluster This cluster changes the execution state of foreground and background PLC programs by modifying i5.

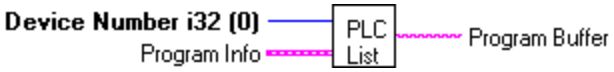
First Time Bool (F) When TRUE this VI determines the enable state of foreground and background PLC programs and sets Output PLC Exec State Cluster to reflect that state.

Output PLC Exec Cluster Display the current execution state of foreground and background PLC programs.

New Output Bool (F) TRUE when First Time is TRUE or either button in Input PLC Exec State Cluster changes.

PmacPLCList

This VI lists the contents of Pmac program buffers and parses the program into an array with the indices of line ends.



Device Number i32 (0)

Program Info

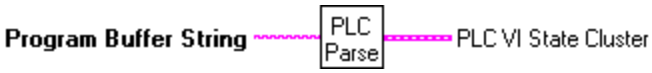
Program Loaded

Program Number i32

Program Buffer

PmacPLCParse

Parse Program Buffer for PLC Number. Provide the information in PLC VI State Cluster. The program load state is not determined by this operation.



Program Buffer String PLC Program buffer to parse for PLC Number.

PLC VI State Cluster Cluster defining currently selected PLC and it execution state.

PmacPLCSelect

Query PMAC for a description of all loaded PLC programs by reading PMAC's internal buffers. If First Time is TRUE Menu String Array contains a sorted list of all loaded PLC programs and their execution state by PLC number for the menu ring in PLC Select Cluster. Button String Array contains information to change the description of the button in PLC Select Cluster so that it toggles the selected PLC's execution state when clicked. The VI maintains New Selection Index as a state from execution to execution. Translation of menu ring selections in PLC Select Cluster into PLC Selected Cluster occurs when First Time Strings is TRUE or either control in PLC Select Cluster changes. New Output TRUE indicates that PLC Selected Cluster, Menu String Array, and Button String Array contain new data.

Device Number i32 (0) Provided by PmacDevOpen

PLC Select Cluster Display loaded PLC programs and their execution state. The execution state can be toggled using the

button.

PLC Select Menu Ring This is the currently selected PLC program and its execution state.

PLC Execute State Bool Toggle the execution state of the currently selected PLC program

First Time Bool (F) When TRUE this VI builds Menu String Array identifying PLC programs loaded in PMAC and their execution state. Button String Array changes the Boolean text in a panel's PLC Select Cluster so that clicking the button will toggle the execution state of the PLC.

New Output Bool (F) TRUE when the menu item or button in PLC Select Cluster change or First Time is TRUE. Menu String Array and Button String Array contain updated data for the application PLC Select Cluster item attribute nodes.

Menu String Array Array of strings identifying PLC programs loaded in PMAC memory and their execution state. Can be used to set Menu Ring attributes.

New Selection Index i32 Last Selection Index State

Button String Array Array of strings for PLC Execute State button in PLC Select Cluster.

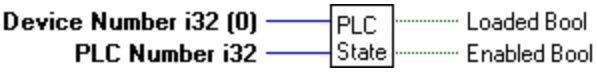
PLC Selected Cluster Cluster defining currently selected PLC and it execution state.

Execute State Bool TRUE indicates that PLC Number is enabled.

PLC Number i32 PLC Number this cluster refers to.

PmacPLCState

Query PMAC for the load state and enable state of PLC Number. If the PLC is loaded set Loaded TRUE. If the PLC is enmabled set Enabled TRUE.



PLC Number i32 PLC Number whose load and enable state is to be queried.

Device Number i32 (0) Provided by PmacDevOpen.

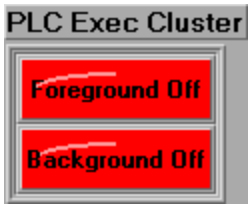
Enabled Bool When TRUE the specified PLC Number is enabled.

Loaded Bool TRUE indicates that the specified PLC number is properly loaded in PMAC.

Indicator and Control Clusters

PmacPLCExec

This cluster enables the execution of foreground and background PLC programs by modifying i5.



PLC Exec Cluster This cluster enables the execution of foreground and background PLC programs by modifying i5.

- **Foreground** Enable or Disable (toggle) foreground PLC program execution.
- **Foreground copy** Enable or Disable (toggle) background PLC program execution.

PmacPLCSelect

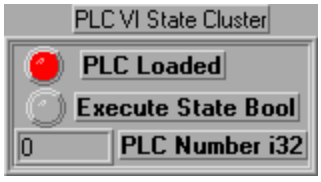
This cluster displays loaded PLC programs and their execution state. The execution state can be toggled using the button.

PLC Select Cluster Display loaded PLC programs and their execution state. The execution state can be toggled using the button.

- **PLC Select Menu Ring** This is the currently selected PLC program and its execution state.
- **PLC Execute State Bool** Toggle the execution state of the currently selected PLC program

PmacPLCVIState

Cluster defining PLC and it execution state.



PLC VI State Cluster Cluster defining PLC and it execution state.

PLC Loaded TRUE when PLC Number is loaded in PMAC memory

Execute State Bool TRUE indicates that PLC Number is enabled.

PLC Number i32 PLC Number this cluster refers to.

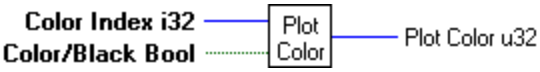
PmacPlot

\PmacPlot - A few generally useful plotting VIs for setting plot colors and legends. An XY Chart buffer is available to make an XY plot into an XY strip chart.

VIs

PmacPlotColors

This VI maintains an array of standard colors for plots that are indexed by Color Index. When Color/Black is TRUE Plot Color is the indexed color. Otherwise Plot Color is Black.



Color/Black Bool When TRUE select the color specified for Color Index. Plot Color is Black otherwise.

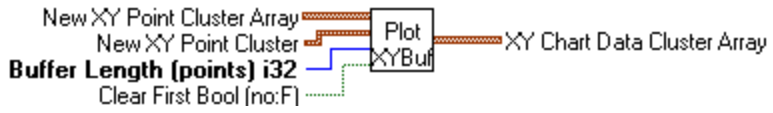
Color Index i32 Index of standard color to use

Plot Color u32 Color for setting colors in charts and plots

PmacPlotXYChartBuffer

This VI uses an XY graph to mimic the behavior of an XY chart, in that it plots a stipulated amount of the most recently acquired data points. The chart data is stored in a static local variable of the XY Chart Data Cluster Array.

Because the VI has been marked reentrant (in VI setup), you can make multiple parallel calls to the VI without confusing the memory. Use the New XY Point Cluster Array input to supply multiple points at once, or leave the array unwired (empty) and use the New XY Point Cluster input to supply one point at a time. If the locally stored chart array is not equal to chart length, it is expanded to that length by replicating the most recently acquired point. Data in excess of the history length is discarded. Set Clear First to clear the chart history before processing the new points.


New XY Point Cluster Array
New XY Point Cluster
Buffer Length (points) i32
Clear First Bool (no:F)
Plot XYBuf
XY Chart Data Cluster Array

Buffer Length (points) i32 Defines size of the circular buffer used to remember points in the chart. Default is 100.

Clear First Bool (no:F) If TRUE, this VI first resets all of the points in the circular buffer to the default value before adding new values from new point or new points.

New XY Point Cluster New point to be added to circular buffer. This control will be used only if the new points array is empty.

- **x**
- **y**

New XY Point Cluster Array New points to be appended to circular buffer. Use this control/input only if adding more than one point at a time. If this array is empty, New XY Point is appended to the buffer.

- **XY Point**
- **x**
- **y**

XY Chart Data Cluster Array The current contents of the circular buffer to be passed to the XY chart.

- **XY Point**
- **x**
- **y**

PmacPQM

\PmacPQM - PMAC program execution is parametrically specified using P, Q, and M variables. For example, the number of times a move is executed, the increment of a move, or the radius of a circular move can all be specified using Ps and Qs. Specific machine inputs and outputs, and internal registers are accessible using M-variables. Mapping of these quantities to LabVIEW controls is facilitated by the ICVs in this collection. In addition, the ability to log this information to a LabVIEW datalog file and re-execute the motion at a later time is provided.

VIs

PmacPQMArray

If Set\Get is FALSE or not wired get the array of PQM Variables specified by Input Variant Array. Response Available will be TRUE to indicate Output Variant Array contains the new values. If Set\Get is TRUE set the array of PQM Variables using

Input Variant Array. Response Available will be FALSE and Output Variant Array defaults to Input Variant Array.

If you specify an M-Variable it must be defined usingPewin32, PmacTerminal, or PmacCommSendString.

Device Number i32 (0) Provided by PmacDevOpen

Set/Get Bool (F) Set array of PQM Variables when TRUE. Get array of PQM Variables when FALSE or unwired.

Input Variant Array Array of Variant Clusters to use for Set operation.

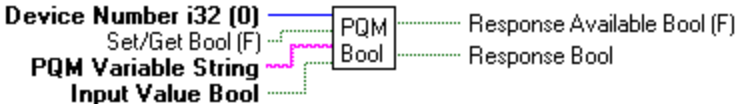
Response Available Bool (F) TRUE when Get operation produces valid Output Variant Array.

Output Variant Array Specified array of type-neutral PQM Variant Clusters for Get operation. Input Variant Array for Set operation.

PmacPQMBool

If Set\Get is FALSE or not wired get the Boolean PQM Variable specified by PQM Variable String. Response Available will be TRUE to indicate Response contains the new value. If Set\Get is TRUE set the Boolean PQM Variable using Input Value. Response Available will be FALSE and Response defaults to Input Value.

If you specify an M-Variable it must be defined usingPewin32, PmacTerminal, or PmacCommSendString.



Device Number i32 (0) Provided by PmacDevOpen

PQM Variable String PQM Variable name. e.g. "P34"

Input Value Bool Value to use for Set operation.

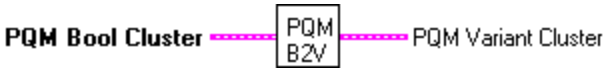
Set/Get Bool (F) Set PQM Variable when TRUE. Get PQM Variable when FALSE or unwired.

Response Bool Specified PQM Variable for Get operation. Input Value for Set operation.

Response Available Bool (F) TRUE when Get operation produces valid Response

PmacPQMBool2Var

Convert the PQM Bool Cluster to a type-neutral PQM Variant Cluster.



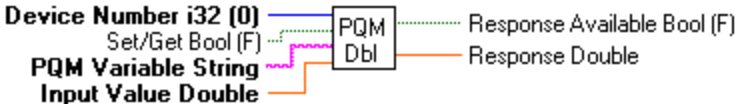
PQM Bool Cluster PQM cluster from control or other operation.

PQM Variant Cluster Type-neutral cluster PQM Bool Cluster is converted to.

PmacPQMDbl

If Set\Get is FALSE or not wired get the Double PQM Variable specified by PQM Variable String. Response Available will be TRUE to indicate Response contains the new value. If Set\Get is TRUE set the Double PQM Variable using Input Value. Response Available will be FALSE and Response defaults to Input Value.

If you specify an M-Variable it must be defined usingPewin32, PmacTerminal, or PmacCommSendString.



Device Number i32 (0) Provided by PmacDevOpen

PQM Variable String PQM Variable name. e.g. "P34"

Input Value Double Value to use for Set operation.

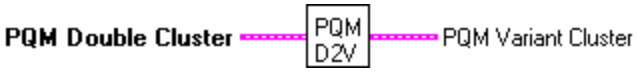
Set/Get Bool (F) Set PQM Variable when TRUE. Get PQM Variable when FALSE or unwired.

Response Double Specified PQM Variable for Get operation. Input Value for Set operation.

Response Available Bool (F) TRUE when Get operation produces valid Response

PmacPQMDbl2Var

Convert the PQM Double Cluster to a type-neutral PQM Variant Cluster.



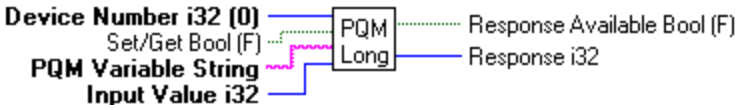
PQM Double Cluster PQM cluster from control or other operation.

PQM Variant Cluster Type-neutral cluster PQM Double Cluster is converted to.

PmacPQMLong

If Set\Get is FALSE or not wired get the Long PQM Variable specified by PQM Variable String. Response Available will be TRUE to indicate Response contains the new value. If Set\Get is TRUE set the Long PQM Variable using Input Value. Response Available will be FALSE and Response defaults to Input Value.

If you specify an M-Variable it must be defined usingPewin32, PmacTerminal, or PmacCommSendString.



Device Number i32 (0) Provided by PmacDevOpen

PQM Variable String PQM Variable name. e.g. "P34"

Input Value i32 Value to use for Set operation.

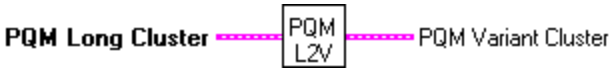
Set/Get Bool (F) Set PQM Variable when TRUE. Get PQM Variable when FALSE or unwired.

Response i32 Specified PQM Variable for Get operation. Input Value for Set operation.

Response Available Bool (F) TRUE when Get operation produces valid Response

PmacPQMLong2Var

Convert the PQM Long Cluster to a type-neutral PQM Variant Cluster.



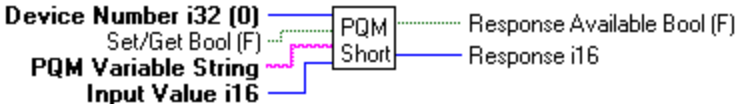
PQM Long Cluster PQM cluster from control or other operation.

PQM Variant Cluster Type-neutral cluster PQM Long Cluster is converted to.

PmacPQMShort

If Set\Get is FALSE or not wired get the Short PQM Variable specified by PQM Variable String. Response Available will be TRUE to indicate Response contains the new value. If Set\Get is TRUE set the Short PQM Variable using Input Value. Response Available will be FALSE and Response defaults to Input Value.

If you specify an M-Variable it must be defined usingPewin32, PmacTerminal, or PmacCommSendString.



Device Number i32 (0) DeviceNumber

PQM Variable String PQM Variable name. e.g. "P34"

Input Value i16 Value to use for Set operation.

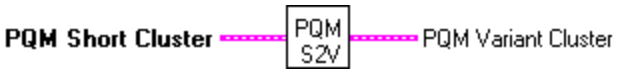
Set/Get Bool (F) Set PQM Variable when TRUE. Get PQM Variable when FALSE or unwired.

Response i16 Specified PQM Variable for Get operation. Input Value for Set operation.

Response Available Bool (F) TRUE when Get operation produces valid Response

PmacPQMShort2Var

Convert the PQM Short Cluster to a type-neutral PQM Variant Cluster.

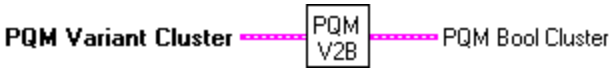


PQM Short Cluster PQM cluster from control or other operation.

PQM Variant Cluster Type-neutral cluster PQM Short Cluster is converted to.

PmacPQMVar2Bool

Convert a type-neutral PQM Variant Cluster to a PQM Bool Cluster.



PQM Variant Cluster Type-neutral cluster to be converted to PQM Bool Cluster.

PQM Bool Cluster PQM cluster for indicator or other operation.

PmacPQMVar2Dbl

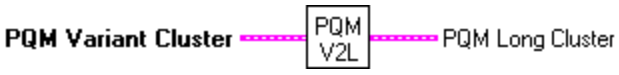
Convert a type-neutral PQM Variant Cluster to a PQM Double Cluster.

PQM Variant Cluster Type-neutral cluster to be converted to PQM Double Cluster.

PQM Double Cluster PQM cluster for indicator or other operation.

PmacPQMVar2Long

Convert a type-neutral PQM Variant Cluster to a PQM Long Cluster.

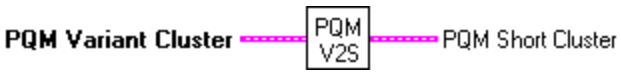


PQM Variant Cluster Type-neutral cluster to be converted to PQM Long Cluster.

PQM Long Cluster PQM cluster for indicator or other operation.

PmacPQMVar2Short

Convert a type-neutral PQM Variant Cluster to a PQM Short Cluster.



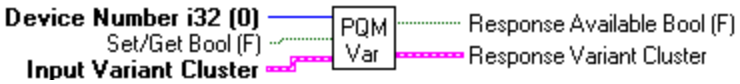
PQM Variant Cluster Type-neutral cluster to be converted to PQM Short Cluster.

PQM Short Cluster PQM cluster for indicator or other operation.

PmacPQMVariant

If Set\Get is FALSE or not wired get the variant PQM Variable specified by PQM Variable String. Response Available will be TRUE to indicate Response contains the new value. If Set\Get is TRUE set the variant PQM Variable using Input Value. Response Available will be FALSE and Response defaults to Input Value.

If you specify an M-Variable it must be defined usingPewin32, PmacTerminal, or PmacCommSendString.



Device Number i32 (0) Provided by PmacDevOpen

Input Variant Cluster Variant Cluster to use for Set operation.

Set/Get Bool (F) Set PQM Variable when TRUE. Get PQM Variable when FALSE or unwired.

Response Available Bool (F) TRUE when Get operation produces valid Response

Response Variant Cluster Specified type-neutral PQM Variant Cluster for Get operation. Input Variant Cluster for Set operation.

Indicator and Control Clusters

PmacPQMBool

Cluster for tying PQM variable definition with a Boolean control/indicator. After inserting on your panel specify a PQM variable name for the Variable Item and make it the default using RightMouseButton»Data Operations»Make Current Value Default. Replace Control to reflect your requirements.

PQM Bool Cluster Cluster for tying PQM variable definition with a Boolean control/indicator. After inserting on your panel specify a PQM variable name for the Variable Item and make it the default using RightMouseButton»Data Operations»Make Current Value Default. Replace Control to reflect your requirements.

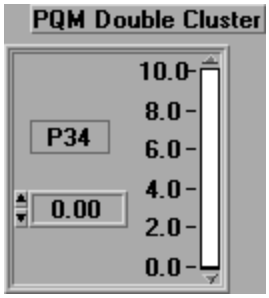
Variable String defining PQM Variable name. e.g. "P34"

Control Control for associated PQM Variable

PQM Type

PmacPQMDbl

Cluster for tying PQM variable definition with a double control/indicator. After inserting on your panel specify a PQM variable name for the Variable Item and make it the default using RightMouseButton»Data Operations»Make Current Value Default. Replace Control to reflect your requirements.



PQM Double Cluster Cluster for tying PQM variable

definition with a double control/indicator. After inserting on your panel specify a PQM variable name for the Variable Item and make it the default using RightMouseButton»Data Operations»Make Current Value Default. Replace Control to reflect your requirements.

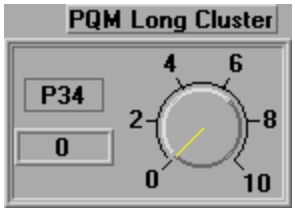
Variable String defining PQM Variable name. e.g. "P34"

Control Control for associated PQM Variable

PQM Type

PmacPQMLong

Cluster for tying PQM variable definition with an i32 control/indicator. After inserting on your panel specify a PQM variable name for the Variable Item and make it the default using RightMouseButton»Data Operations»Make Current Value Default. Replace Control to reflect your requirements.



PQM Long Cluster Cluster for tying PQM variable definition with an i32 control/indicator. After inserting on your panel specify a PQM variable name for the Variable Item and make it the default using RightMouseButton»Data Operations»Make Current Value Default. Replace Control to reflect your requirements.

Variable String defining PQM Variable name. e.g. "P34"

Control Control for associated PQM Variable

PQM Type

PmacPQMShort

Cluster for tying PQM variable definition with an i16 control/indicator. After inserting on your panel specify a PQM variable name for the Variable Item and make it the default using RightMouseButton»Data Operations»Make Current Value Default. Replace Control to reflect your requirements.

PQM Short Cluster Cluster for tying PQM variable

definition with an i16 control/indicator. After inserting on your panel specify a PQM variable name for the Variable Item and make it the default using RightMouseButton»Data Operations»Make Current Value Default. Replace Control to reflect your requirements.

Variable String defining PQM Variable name. e.g. "P34"

Control Control for associated PQM Variable

PQM Type

PmacPQMType

Enumerated type defining supported PMACPanel types. Defined values are Bool, Short, Long, Double.

PQM Type Enumerated type defining supported PMACPanel types. Defined values are Bool, Short, Long, Double.

PmacPQMVariant

Cluster for tying PQM variable definition with a PQM types-neutral string. This cluster is generally not used on application panels.

PQM Variant Cluster Cluster for tying PQM variable definition with a PQM types-neutral string. This cluster is generally not used on application panels.

Variable String defining PQM Variable name. e.g. "P34"

Control Type-neutral string containing PQM name and control/indicator value.

PQM Type

PmacProgram

\PmacProgram - This collection provides tools at a variety of levels to edit, download, debug, monitor, and encapsulate motion programs. Encapsulated programs

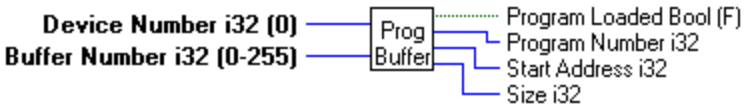
- Load themselves when executed

- Know their coordinate system and program number
- Can be executed by the click of a button
- Indicate the state of their execution
- Can be modified, monitored, and debugged from a powerful front panel
- Accept P, Q, and M variant data types from the **\PmacPQM** tools.

VIs

PmacProgBuffer

PMAC maintains a list of program buffers in memory. The list identifies the program number and start address of the program. Buffer Number is the program buffer entry queried for program existence. If Program Loaded is TRUE the buffer points to Program Number, of Size words, located at Start Address. If Program Loaded is FALSE the buffer does not point to a motion program.



Device Number i32 (0) Provided by PmacDevOpen

Buffer Number i32 (0-255) Program buffer between 0 and 255 to fetch information for.

Start Address i32 Start address of program buffer in PMAC memory

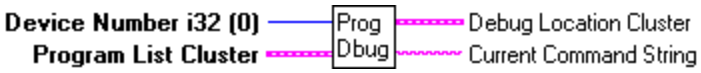
Size i32 Size of program buffer.

Program Loaded Bool (F) TRUE when Buffer Number contains a program.

Program Number i32 Program Number contained in Buffer Number

PmacProgDebug

Query PMAC for currently executing line in Coord Number and output the response in Current Command. Determine the scroll position and characters that delimit this line in List Buffer and create Debug Location Cluster for setting selection attributes in a multi-line string control for real-time display of Program Number's execution. This information is obtained from PMAC using the LIST PE command.



Device Number i32 (0) Provided by PmacDevOpen

Program List Cluster Cluster specifying a Program Number, Coord Number, Program Load state, Program Listing, Index

of line indexes, and a Boolean indicating the existence of Program Listing.

List Buffer String Program Listing of loaded program

Line Selection Index Array Array indicating which character in Program List String new lines begin on.

Program Listed Bool TRUE when listed.

Program VI State Cluster This cluster identifies whether Program Number is loaded in PMAC and which Coord Number to run Program Number in.

Debug Location Cluster Cluster of information for string control attributes. The items define the Scroll Position of the string in the buffer, and the Start and End Character of the line currently executing.

Selection Start and End character in List Buffer for currently executing program line.

Character Start

Character End

Scroll Position Number of currently executing line in List Buffer.

Current Command String PMAC response to query for currently executing program line in Coord Number.

PmacProgDownLoadBuffer

Write Program Buffer to a temporary file then compile and download to PMAC. Down Load File Path and Down Load Log File Path specify the temporary file's name and compile log. If a compile error occurs a modal pop-up panel containing the log is displayed. Program VI State Cluster contains identifies Program Loaded, its Program Number, and Coord Number to run the program in.

Program Buffer String Program buffer to down load and parse for Program Number and Coord Number to run the program in.

DeviceNumber i32 (0) Provided by PmacDevOpen

Down Load File Path Path of temporary file used to down load Program Buffer.

Down Load Log File Path Path of file containing down load

compile log.

Program VI State Cluster This cluster identifies whether Program Number is loaded in PMAC and which Coord Number to run Program Number in.

PmacProgEdit

Manage common editing operations on Input Buffer String as specified by Program Edit Cluster. Input File Path is the default file path to use for Load, Save, or Save As operations. New Output Buffer is TRUE when a Load or Clear Window operation puts new data in Output Buffer. New Path is TRUE when a Load, Save, or Save As operation modifies the file path.

Load - Load a file into Output Buffer.

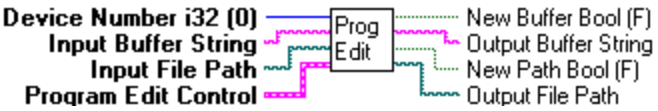
Save - Save Input Buffer to Input File Path.

Save As - Query the user for a new file to save Input Buffer to.

Clear Window - Put an empty string in Output Buffer.

Down Load - Compile and down load Input Buffer to PMAC.

Show Log - Display the contents of the compile log.



Program Edit Control Cluster of latched buttons to specify buffer operations performed by PmacProgEdit.

Input Buffer String Text buffer to apply edit operations to.

Input File Path File path for edit operations.

Device Number i32 (0) Provided by PmacDevOpen

Output Buffer String New buffer data as the result of Clear Window or Load operations.

New Buffer Bool (F) When TRUE this indicates that Output Buffer contains new data as the result of a Load operation or Clear Window.

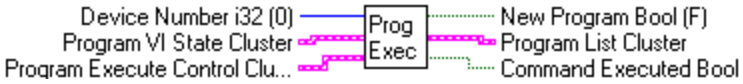
Output File Path New Output File Path as the result of a Load, Save, or Save As operation.

New Path Bool (F) When TRUE this indicates that Output File Path contains new data as the result of a Load, Save, or Save As operation.

PmacProgExec

Interactively execute the program specified in Program VI State Cluster in response to button clicks in Program Execute Control Cluster. New Program is TRUE when

Begin is clicked and Program List Cluster contains a new listing. Command Executed is TRUE when any button in Program Execute Control Cluster is clicked.



Device Number i32 (0) Provided by PmacDevOpen

Program Execute Control Cluster Cluster of latched buttons to interactively control the execution of motion programs.

Program VI State Cluster This cluster identifies whether Program Number is loaded in PMAC and which Coord Number to run Program Number in.

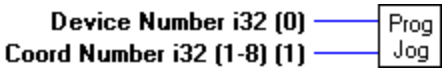
New Program Bool (F) TRUE when a new Program List Cluster is generated in response to clicking Begin in Program Execute Cluster.

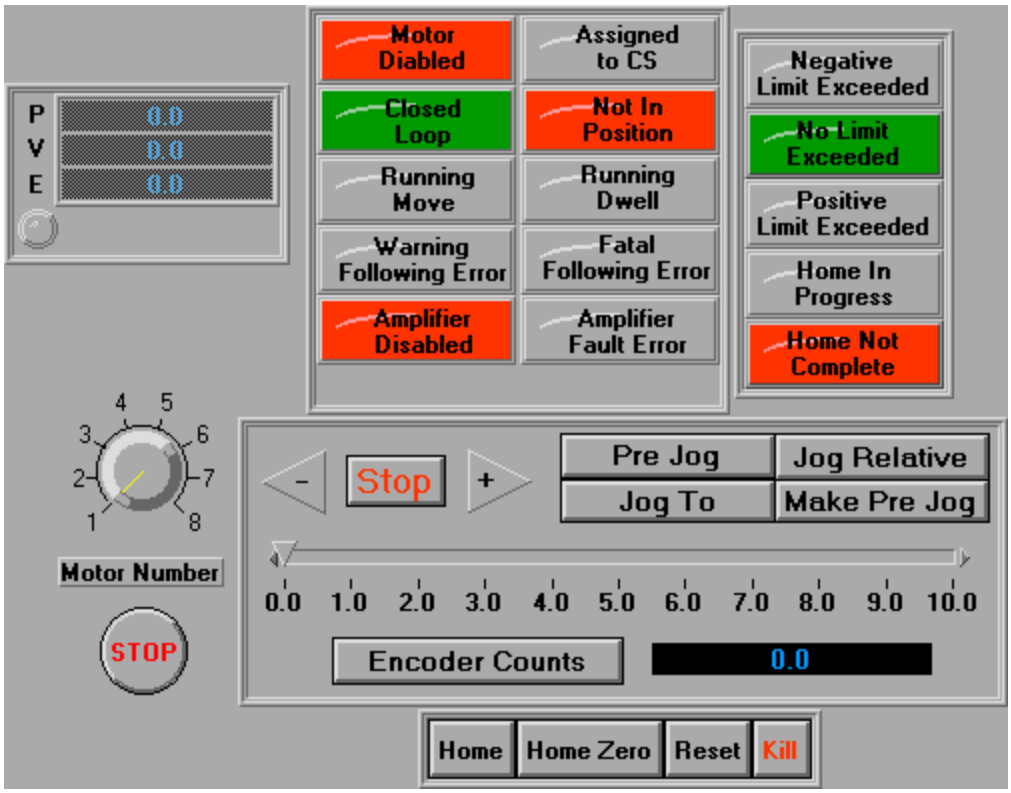
Command Executed Bool TRUE when any Program Execute Control Cluster button is clicked

Program List Cluster Cluster specifying a Program Number, Coord Number, Program Load state, Program Listing, Index of line indexes, and a Boolean indicating the existence of Program Listing.

PmacProgJog

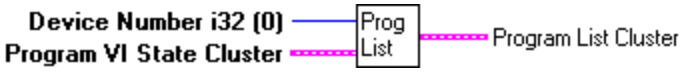
This is a modal pop-up dialog that allows Jogging in the current coordinate system. It executes a return to the pre-jog position when Stop is clicked.





PmacProgList

Query PMAC for a listing of Program Number contained in Program VI State Cluster. Parse the listing to locate line starts in the listing. Bundle all of this information in Program List Cluster.



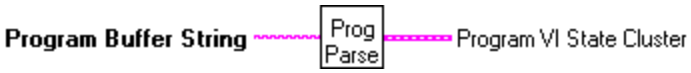
Device Number i32 (0) Provided by PmacDevOpen

Program VI State Cluster This cluster identifies whether Program Number is loaded in PMAC and which Coord Number to run Program Number in.

Program List Cluster Cluster specifying a Program Number, Coord Number, Program Load state, Program Listing, Index of line indexes, and a Boolean indicating the existence of Program Listing.

PmacProgParse

Parse Program Buffer to Program Number and Coord Number to run the program in. Provide the information in Program VI State Cluster. The program load state is not determined by this operation.

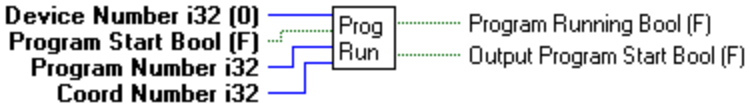


Program Buffer String Program buffer to parse for Program Number and Coord Number to run the program in.

Program VI State Cluster This cluster identifies whether Program Number is loaded in PMAC and which Coord Number to run Program Number in.

PmacProgRun

Control and monitor the execution of Program Number in Coord Number. The specified program is started when Program Start is TRUE and no program is currently running in Coord Number. Program Running indicates that some program - maybe not Program Number - is running in Coord Number. Output Program Start is a copy of Program Start and can be used to sequence program execution with other operations.



Device Number i32 (0) Provided by PmacDevOpen

Coord Number i32 CS to run the program in.

Program Number i32 Program Number to run.

Program Start Bool (F) Start Program Number in Coord Number when TRUE as long a there is no program executing in Coord Number.

Program Running Bool (F) TRUE when ANY program is running Coord Number.

Output Program Start Bool (F) Copy of Program Start. Can be used to sequence operations.

PmacProgSelect

Query PMAC for a description of all loaded motion programs by reading PMAC's internal buffers. If First Time is TRUE Menu String Array contains a sorted list of all loaded programs by program number. The VI maintains New Selection Index as a state from execution to execution. Translation of Program Selection Index into Program Number occurs when First Time Strings is TRUE or Program Selection Index is not equal to New Selection Index. New Output TRUE indicates that Program Number, New Selection Index, and Menu String Array contain new data.

Device Number i32 (0) Provided by PmacDevOpen

Program Selection Index Index into Menu String Array and corresponds to a program buffer. This is translated into Program Number.

First Time Bool (F) When TRUE this VI builds Menu String Array identifying motion programs loaded in PMAC.

Menu String Array Array of strings identifying motion programs loaded into PMAC memory. Can be used to set Menu Ring attributes.

New Selection Index i32 Copy of Program Selection Index

New Output Bool (F) TRUE when Program Selection Index changes from the last execution or First Time is TRUE. Program Number is the translated index, New Selection is a copy of Program Selection Ring, and Menu String Array contains an updated listing of programs in memory.

Program Number i32 Program Number translated from Program Selection Index.

Indicator and Control Clusters

PmacProgEdit

Cluster of latched buttons to specify buffer operations performed by PmacProgEdit.

Program Edit Control Cluster of latched buttons to specify buffer operations performed by PmacProgEdit.

- **Load** Load program buffer from a file
- **Save** Save program buffer to a file
- **Save As** Save a program buffer to a file
- **Clear Window** Clear a program buffer
- **Down Load** Write program buffer to a temp file, compile the contents, and down load to PMAC. Compile errors display a log file.
- **Show Log** Display the compile log for the down

load.

PmacProgExec

Cluster of latched buttons to interactively control the execution of motion programs.

Program Execute Control Cluster Cluster of latched buttons to interactively control the execution of motion programs.

Begin The command sent to PMAC is &*CS*b*PROG* where *CS* is the specified coordinate system number and *PROG* is the current program number.
This command causes PMAC to set the program counter of the addressed coordinate system to the specified motion program and location. It is usually used to set the program counter to the Beginning of a motion program. The next R or S command will start execution at this point.
If {constant} is an integer, the program counter will point to the beginning of the program whose number matches {constant}. Fixed motion program buffers (PROG) can have numbers from 1 to 32,767. The rotary motion program carries program number 0 for the purpose of this command.
If {constant} is not an integer, the fractional part of the number represents the line label (N or O) in the program to which to point. The fractional value multiplied by 100,000 determines the number of the line label to which to point (it fills the fraction to 5 decimal places with zeros).
If a motion program buffer (including ROTARY) is open when this command is sent to PMAC, the command is entered into the buffer for later execution, to be interpreted as a B-axis move command.

Run The command sent to PMAC is R.
This command causes the addressed PMAC coordinate system to start continuous execution of the motion program addressed by the coordinate system's program counter from the location of the program counter. Addressing of the program counter is done initially using the B{constant} command. The coordinate system must be in a proper condition in order for PMAC to accept this command.

Otherwise PMAC will reject this command with an error; if I6 is 1 or 3, it will report the error number.
The following conditions can cause PMAC to reject this command (also listed are the remedies):
-Both limits set for a motor in coordinate system (ERR010); clear limits
-Another move in progress (ERR011); stop move (e.g. with J/)
-Open-loop motor in coordinate system (ERR012); close loop with J/ or A
-Inactivated motor in coordinate system (ERR013); change Ix00 to 1 or remove motor from coordinate system
-No motors in the coordinate system (ERR014); put at least 1 motor in C.S.
-Fixed motion program buffer open (ERR015); close buffer and point to program
-No program pointed to (ERR015); point to program with B command

Step The command sent to PMAC is S.
This command causes the addressed PMAC coordinate system to start single-step execution of the motion program addressed by the coordinate system's program counter from the location of the program counter. Addressing of the program counter is done initially using the B{constant} command.
Single-step execution processes all lines down to and including the next move command, DWELL, DELAY; or if a BLOCKSTART statement is encountered in the program, all lines down to and including the next BLOCKSTOP statement.
If the coordinate system is already executing a motion program when this command is sent, the command puts the program in single-step mode, so execution will stop at the end of the currently executing move. In this case, its action is the equivalent of the Q command.
The coordinate system must be in a proper condition in order for PMAC to accept this command. Otherwise PMAC will reject this command with an error; if I6 is 1 or 3, it will report the error number.
The same conditions that cause PMAC to reject an R command will cause to reject an S command; refer to those conditions under the R command specification:

Program Hold The command sent to PMAC is \.
This command causes PMAC to do a program hold of the currently addressed coordinate system in a manner that permits jogging of the motor in the coordinate system while in hold mode, provided PMAC is in segmented mode (LINEAR or CIRCLE mode with I13 > 0). If PMAC is in segmentation mode (I13 = 0, or other move mode), the \ command has the same effect as the H command, bringing the motors to a stop in the same way, but not permitting

any moves while in feed hold mode.
The rate of deceleration to a stop in feed hold mode, and from a stop on the subsequent R command, is controlled by I52. This is a global I-variable that controls the rate for all coordinate systems.
Once halted in hold mode, program execution can be resumed by the R command. In the meantime, the individual motor may be jogged away from this point, but they must be returned to this point using the J= command before program execution may be resumed. An attempt to resume program execution from a different point will result in an error (ERR017 reported if I6 = 1 or 3). If resumption of this program from this point is not desired, the A (Abort) command should be issued before other programs are run.

Feed Hold The command sent to PMAC is H.
This causes the currently addressed coordinate system to suspend execution of the program starting immediately by bringing its time base value to zero, decelerating along its path at a rate defined by the coordinate system I-variable Ix95. Technically the program is still executing after an H command, but at zero speed.
The H command is very similar in effect to a %0 command, except that deceleration is controlled by Ix95, not Ix94, and execution can be resumed with an R or an S command, instead of a %100 command. In addition, H works under external time base, whereas a %0 command does not
Full speed execution along the path will commence again on an R or S command. The ramp up to full speed will also take place at a rate determined by Ix95 (full time-base value, either internally or externally set). Once the full speed is reached, Ix94 determines any time-base changes.

Abort The command sent to PMAC is A.
This command causes all axes defined in the current coordinate system to begin immediately to decelerate to a stop, aborting the currently running motion program (if any). It also brings any disabled (killed) or open-loop motors (defined in the current coordinate system) to an enabled zero-velocity closed-loop state.
If moving, each motor will decelerate its commanded profile at a rate defined by its own motor I-variable Ix15. If there is significant following error when the A command is issued, it may take a long time for the actual motion to stop. Although the command trajectory is brought to a stop at a definite rate, the actual position will continue to "catch up" to the commanded position for a longer time.
Note that a multi-axis system may not stay on its programmed path during this deceleration.

Abort commands are not meant to be recovered from gracefully. If you wish to resume easily, us the H or Q command instead.
Motion program execution may resume (if a motion program was in fact aborted) by issuing either an R or S command, but two factors must be considered. First, the starting positions for calculating the next move will be the original end positions of the aborted move unless the PMATCH command is issued or I14=1. Second, the move from the aborted position to the next move end position may not be possible or desirable. The J= command may be used to jog each motor in the coordinate system to the original end position of the aborted move, provided I13 is 0 (no segmentation mode).

Halt "Q" The command sent to PMAC is Q.
This causes the currently addressed coordinate system to cease execution of the program at the end of the currently executing move or the next move if that has already been calculated. The program counter is set to the next line in the program, so execution may be resumed at that point with an R or S command..

Halt "/" The command sent to PMAC is /.
This command causes PMAC to halt the execution of the motion program running in the currently addressed coordinate system at the end of the currently executing move, provided PMAC is in segmentation mode (I13 > 0). If PMAC is not in segmentation mode (I13 = 0), the / command has the same effect as the Q command, halting execution at the end of the latest calculated move, which can be 1 or 2 moves past the currently executing move. Once halted at the end of the move, program execution can be resumed with the R command. In the meantime, the individual motors may be jogged away from this point, but they must all be returned to this point using the J= command before program execution may be resumed. As attempt to resume program execution for a different point will result in an error (ERR017 reported if I6 = 1 or 3). If resumption of the program from this point is not desired, the A (Abort) command should be issued before other programs are run.

Jog Dialog Pop a jog panel up to allow motor jogging after a Prog Hold. The jog panel must be closed before the panel before you can return to this panel to restart the program with Run or Step. Closing the panel returns all jogged motors to their pre-jog position with j=. Commands sent to PMAC are generated from the panel.

PmacProgList

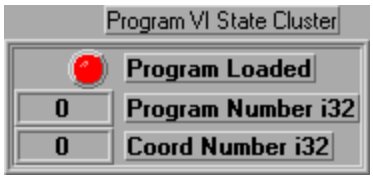
Cluster specifying a Program Number, Coord Number, Program Load state, Program Listing, Index of line indexes, and a Boolean indicating the existence of Program Listing.

Program List Cluster specifying a Program Number, Coord Number, Program Load state, Program Listing, Index of line indexes, and a Boolean indicating the existence of Program Listing.

- **List Buffer String** Program Listing of loaded program
- **Line Selection Index Array** indicating which character in Program List String new lines begin on.
- **Program Listed Bool** TRUE when List Buffer contains a program listing.
- **Program VI State Cluster** This cluster identifies whether Program Number is loaded in PMAC and which Coord Number to run Program Number in.

PmacProgVIState

This cluster identifies whether Program Number is loaded in PMAC and which Coord Number to run Program Number in.



Program VI State Cluster This cluster identifies whether Program Number is loaded in PMAC and which Coord Number to run Program Number in.

- **Program Loaded** TRUE when Program Number is loaded in PMAC memory

Program Number i32 Identifies a program number loaded whose load state is identified by Program Loaded.

Coord Number i32 CS to run Program Number in.

PmacResponse

\PmacResponse - Send formatted command strings to PMAC and convert ASCII response strings into numerical values for use in you PMACPanel application.

VIs

PmacRespGetBool

If Command String is not the empty string send it to PMAC and wait for a response. If Response Available is TRUE Response contains a valid response. Otherwise Response is FALSE.

Device Number i32 (0) Provided by PmacDevOpen

Command String PMAC on-line command string to send to PMAC

Response Available Bool (F)) TRUE when Response is valid

Response Bool (F) Boolean response to Command String. If the response is not valid Response = FALSE.

PmacRespGetDbl

If Command String is not the empty string send it to PMAC and wait for a response. If Response Available is TRUE Response contains a valid response. Otherwise Response is 0.0.

Device Number i32 (0) Provided by PmacDevOpen

Command String PMAC on-line command string to send to PMAC

Response Available Bool (F) TRUE when Response is valid

Response Double (0.0) Double response to Command String. If the response is not valid Response = 0.0.

PmacRespGetLong

If Command String is not the empty string send it to PMAC and wait for a response. If Response Available is TRUE Response contains a valid response. Otherwise Response is 0.

Device Number i32 (0) Provided by PmacDevOpen

Command String PMAC on-line command string to send to PMAC

Response Available Bool (F) TRUE when Response is valid

Response i32 (0) Long response to Command String. If the response is not valid Response = 0.

PmacRespGetShort

If Command String is not the empty string send it to PMAC and wait for a response. If Response Available is TRUE Response contains a valid response. Otherwise Response is 0.

Device Number i32 (0) Provided by PmacDevOpen

Command String PMAC on-line command string to send to PMAC

Response Available Bool (F) TRUE when Response is valid

Response i16 (0) Short response to Command String. If the response is not valid Response = 0.

PmacRespGetULong

If Command String is not the empty string send it to PMAC and wait for a response. If Response Available is TRUE Response contains a valid response. Otherwise Response is 0.

Device Number i32 (0) Provided by PmacDevOpen

Command String PMAC on-line command string to send to PMAC

Response Available Bool (F) TRUE when Response is valid

Response u32 (0) ULong response to Command String. If the response is not valid Response = 0.

PmacRespGetUShort

If Command String is not the empty string send it to PMAC and wait for a response. If Response Available is TRUE Response contains a valid response. Otherwise Response is 0.

Device Number i32 (0) Provided by PmacDevOpen

Command String PMAC on-line command string to send to PMAC

Response Available Bool (F) TRUE when Response is valid

Response u16 (0) Ushort response to Command String. If the response is not valid Response = 0.

PmacSubVI

\PmacSubVI - This collection provides support to **PmacProg** and **PmacPLC**.

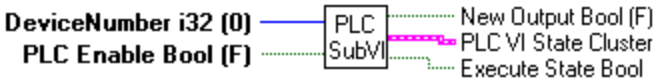
VIs

PmacPLCSubVI

PmacPLCSubVICreate makes a copy of this VI with a new name that matches the name of a PLC program. Because the PLC program has the same name (with a different extension) this VI knows how to open, parse, load, and run a PLC program without intervention or extra inputs. It allows you to interactively monitor and change the PLC program's execution state. Details of its implementation are contained in the manual.

The VI downloads the associated PLC program when first loaded unless this option is disabled in the diagram and a default for PLC Number are provided for the PLC VI State Cluster.

The VI queries PMAC for the PLC's execution state every execution. This is done whether the program is executing or not. New Output is TRUE any time PLC Enable is TRUE.



DeviceNumber i32 (0) Provided by PmacDevOpen

PLC Enable Bool (F) Toggle the PLC program execution state specifed in PLC VI State Cluster.

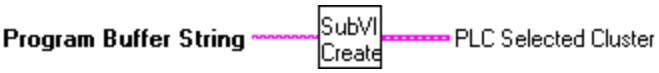
PLC VI State Cluster This cluster contains the PLC VI State information parsed from the program file during initial down load or defined by the VI defaults.

New Output Bool (F) TRUE when Output PQM Variant Array contains new data.

Execute State Bool TRUE indicates that PLC Number is enabled.

PmacPLCSubVICreate

Prompt the user for a file to save Program Buffer to. Make a copy of PmacPLCSubVI.vi changing the name to the same base as the saved program. For example, if Program Buffer is saved to PmacTestPLC1.pmc a new VI named PmacTestPLC1.vi is created from PmacPLCSubVI.



Program Buffer String Program buffer to create encapsulated motion program VI from.

PLC Selected Cluster Cluster defining currently selected PLC and it execution state.

PmacProgSubVI

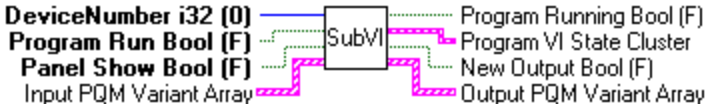
PmacProgSubVICreate makes a copy of this VI with a new name that matches the name of a motion program. Because the motion program has the same name (with a different extension) this VI knows how to open, parse, load, and run a motion program without intervention or extra inputs. It allows you to edit the associated program and interactively excute the program. Details of its implementation are contained in the manual.

The VI downloads the associated program when first loaded unless this option is disabled in the diagram and defaults for Program Number and Coord Number are provided for the Program VI State Cluster.

The interactive panel can be opened and used by setting Panel Show (latched) TRUE. See the documentation for PmacTerminalEdit and PmacTerminalExecute for details on interactive execution. The panel is closed by clicking the Stop button on the panel.

When the latched input Program Run is TRUE Input PQM Variant Array is sent to PMAC to initialize a program's P, Q, or M variables. The program is then started as long as there is no program executing in the associated CS. When Program Running is TRUE this or another program is executing in the associated CS.

The VI queries PMAC for Output PQM Variant Array data every execution to allow monitoring of these variables by your program. This is done whether the program is executing or not. New Output is TRUE when this data is valid.



DeviceNumber i32 (0) Provided by PmacDevOpen

Program Run Bool (F) Start the program specied in

Panel Show Bool (F) During normal operation the panel for this VI is not displayed. When this input is TRUE the panel is opened and displayed.

Input PQM Variant Array The contents of this PQM Variant Array are written to PMAC when Program Run is TRUE prior to actually starting the associated motion program.

Program Running Bool (F) TRUE when the associated PMAC motion program is running in the associated coordinate system.

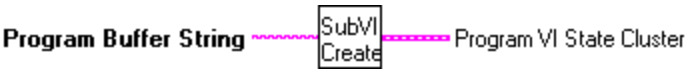
Program VI State Cluster This cluster contains the Program VI State information parsed from the program file during initial down load or defined by the VI defaults.

New Output Bool (F) TRUE when Output PQM Variant Array contains new data.

Output PQM Variant Array This output contains the current values of the PQM values specified by Input PQM Variant Array when New Output is TRUE.

PmacProgSubVICreate

Prompt the user for a file to save Program Buffer to. Make a copy of PmacProgSubVI.vi changing the name to the same base as the saved program. For example, if Program Buffer is saved to PmacTest1.pmc a new VI named PmacTest1.vi is created from PmacProgSubVI.



Program Buffer String Program buffer to create encapsulated motion program VI from.

Program VI State Cluster This cluster identifies whether Program Number is loaded in PMAC and which Coord Number to run Program Number in.

Appendix A - Compiling CINs Using Microsoft Visual C++

Introduction

The following application note on the use of Microsoft Visual C++ for CIN development is available from www.natinst.com. To locate the document, navigate your way through the site by selecting the following links.

Technical Support

Knowledge Base

You can then search for the keyword "CIN" and select the document from the search response.

How do I Compile a Code Interface Node (CIN) using Visual C++ 5.0?

Product Group: LV Software

Product Name: LV-Win95

(Ver / Rev.): 4.1

Problem: How can I compile a code interface node (CIN) from LabVIEW 4.x or 5.x using Visual C++ version 5.0 for Windows 95/NT?

Solution: First a quick overview. When you load a VI in LabVIEW, it automatically takes the G (graphical) source code and compiles it into machine code (assembly language) so that when you press the white Run Arrow you are ready to execute a COMPILED program. Labview.exe was written in C, but the VIs you create NEVER involve a C layer: you compose in G, and that G source code is compiled for you directly and automatically into assembly code. This is why LabVIEW is so fast.

When you create a CIN, LabVIEW is actually absorbing the object code (.lsb) of the C program and assimilating it directly into the existing machine code. What we are doing here is tricking the native C compiler into thinking it's creating a C executable from a .mak file, when in fact we're getting it to create object code that LabVIEW can directly import and incorporate into the vi's own machine code. One fundamental difference between using a CIN in LabVIEW and making a DLL call from the Call

Library Function is that the CIN will operate WITHIN LabVIEW's memory space (which is DYNAMIC), while a DLL call from LabVIEW will operate in it's own STATIC memory space external to LabVIEW's. This is why you can successfully pass static pointers back and forth between LabVIEW and DLLs, but NOT between LabVIEW and CINs.

Here is a step-by-step instructions of how to create a CIN from Visual C++ 5.0:

1. Open a new VI in LabVIEW. Insert the Code Interface Node function from the "Advanced" Sub-palette of the Functions Palette onto your block
diagram.
2. Let's say you want to write a CIN for a simple operation such as A * B = C, you need two inputs and one output. Resize your Code Interface Node function to have three inputs and outputs. Create two digital controls (labels A,B) and wire to the first two inputs (white blocks on the left side of the Code Interface Node function). Create a digital indicator(label C) and wire it to the third output (the third white block on the right side of the Code Interface Node Function). Notice how these blocks changes to show the appropriate data type.
3. Now right-click on the CIN in your block diagram and choose "Create .c File...". This will pop up a window asking you where to save this .c file. Give it a name "mult.c" and save it in a known directory. Now, launch Visual C++ 5.0 and goto File >> New. From the Projects tab in the window that pops up, select Win32 Console Application. Also give a name to the project & the location in the spaces given at the right of the window. When you say OK, it creates a new workspace. On the left side you should see an explorer like window which will have your "ProjectName classes". At the bottom of that window you will see three tabs - Class View, Files View, Info View. Select "Files View". Now you will see "ProjectName Files". Right click on the "ProjectName files" and select add projects to the file. Now select the mult.c file which was created by LabVIEW. This will add mult.c to your project.
4. Open the mult.c by double-clicking on its name under the Files View in the Workspace window. You will notice that LabVIEW has generated a C code with all the required headers. Add your code in the area where it says "/* ENTER YOUR CODE HERE */". Type *C = *A * *B; which will multiply A & B and assign it to C. Go ahead and save your project.
5. The next step is to create a Make file. The .mak file will convince the C compiler to create Labview object code (.lsb) from the C source code you just finished. Open Notepad on your machine and create a new text file. Type the following in that file:

   name=mult (This would be the name of your .c file*)*

   IDE = VC

   type=CIN

   cinlibraries = Kernel32.lib

   CINTOOLSDIR = C:\LabVIEW\cintools (Here you need to provide the path to the cintools directory which is present under you LabVIEW directory)

   !include <$(CINTOOLSDIR)\ntlvsb.mak>

Save this file as Mult.mak file, preferably in the same directory as your mult.c file.

NOTE 1: In the first line, you should provide the name of your .c code

NOTE 2: In the fifth line, " CINTOOLSDIR = C:\LabVIEW\cintools", you should give the path to the "cintools" directory on your computer. It might be different than what is shown here. "cintools" directory is available under your LabVIEW directory. Let's say, if the actual path is C:\Program Files\National Instuments\LabVIEW\cintools, **you should type** C:\Progra~1\Nation~1\LabVIEW\cintools. This is the DOS convention for any names that has white spaces in it. In this case Program Files and National Instruments has white spaces. Therefore, according to the convention, you type the first 6 characters followed by ~1. **This is a very important step.**

NOTE 3: While creating the .mak file, please stick to the uppercase and lowercase as shown above.

6 Now go back to Visual C++ environment and add this make file to your project. You can add this make file by right-clicking on the "ProjectName files" in the "File View" and selecting "Add files to project".

NOTE: You might run into two dialog boxes when you try to load in the mult.mak file. First, MS Vis C/C++ will complain that "This makefile was not generated by Developer Studio". Answer "YES" to "Do you want to continue?", then in the second pop-up dialog agree that Win32 is the Platform you wish to operate on.

Once you add the make file to the project, you should have the .c file and the .mak file in your project. Now goto to the Build menu >> Select "Build ProjectName.exe". This should compile your file successfully. You will get the following message in the Error Window if the executable is combined successfully:

```
--------------------Configuration: mult_ex - Win32 Debug--------------
-----
Microsoft (R) Program Maintenance Utility Version 1.62.7022
Copyright (C) Microsoft Corp 1988-1997. All rights reserved.
mult.c
Microsoft (R) 32-Bit Incremental Linker Version 5.00.7022
Copyright (C) Microsoft Corp 1992-1997. All rights reserved.
LabVIEW resource file, type 'CIN ', name 'mult.lsb', created properly.
mult.exe - 0 error(s), 0 warning(s)
```

NOTE 1: You might run into different errors like "fatal error U1073", "fatal error: C1083 - cannot include extcode.h file", "LNK 2001, LNK 2100 - unresolved external symbol - main". Most of the times these errors happen because the Visual C++ is not able to find a specific Header (.h) file or an Object (.obj) file. Error window will tell you the name of the file it could not find or where the error happened. GO AHEAD A DO A SEARCH FOR THAT FILE IN YOUR COMPUTER AND ADD THAT FILE TO YOUR PROJECT AS

DISCUSSED IN PREVIOUS STEPS. Alternatively, you can add the path of the directory in which you find this file to your project. To do this, go to Tools menu >> Options... >>Select "Directories" tab. Let's say, you want to add the path to the extcode.h file. First you do a search for that file in your computer. You will find this file at C:\Program Files\National Instruments\LabVIEW\cintools\extcode.h. So under the "Directories" tab, double-click on the dotted box at the bottom of the list. This allow you to browse and select the directory. You should include only the path ie., C:\Program Files\National Instruments\LabVIEW\cintools. Do the same for any other files that needs to be included. This will create the search path and Visual C++ will know where to look for those missing files.

7. Now that you've got the object code (mult.lsb), go back to your LabVIEW VI and you would notice that VI is ready to run (The run arrow on top is not broken anymore). If you created your .lsb file before actually building your LabVIEW code, then right-click on the CIN in your block diagram, and choose "Load Code Resource...". Now choose "mult.lsb" from the your file list. You should get a white run arrow, and you can test the VI by multiplying two numbers.

Glossary of Terms

Index

D

Development Tools · 86
DPR
  VIs to access · 37

E

Encoders · 67

H

Homing · 117

I

I-Variables
  Specific
    Ix01 · 59, 160, 176
    Ix02 · 129, 151, 160
    Ix03 · 168
    Ix05 · 148, 180
    Ix06 · 59, 176
    Ix07 · 120
    Ix08 · 96, 119, 120, 157, 158, 159
    Ix09 · 57, 60, 150, 157, 181
    Ix11 · 29, 35, 63, 65, 121, 122, 130, 152, 161, 162, 172, 177
    Ix12 · 29, 35, 63, 65, 122, 162, 172, 177
    Ix13 · 59, 61, 173, 176
    Ix14 · 59, 60, 123, 163, 173, 176
    Ix15 · 104, 123, 162, 163, 207
    Ix16 · 111, 124, 163
    Ix17 · 23, 24, 124, 164
    Ix19 · 125, 126, 154, 164
    Ix20 · 125, 126, 154, 164
    Ix21 · 125, 126, 154, 164
    Ix22 · 111, 124, 125, 127, 155, 163, 164
    Ix23 · 127, 155
    Ix25 · 71, 79, 83, 85, 121, 129, 130, 131, 132, 144, 145, 152, 153, 161
    Ix26 · 74, 120, 127, 155
    Ix28 · 28, 35, 63, 65, 128, 155, 171, 176
    Ix29 · 160, 169
    Ix30 · 120, 157, 158, 159
    Ix31 · 157, 158
    Ix32 · 158
    Ix33 · 158
    Ix34 · 58, 158, 160, 175
    Ix35 · 159
    Ix36 · 159
    Ix37 · 159
    Ix38 · 159
    Ix39 · 160
    Ix63 · 158
    Ix69 · 158, 160
    Ix79 · 160, 169
    Ix80 · 104, 169
    Ix87 · 23, 24
    Ix88 · 23, 24
    Ix89 · 24
    Ix90 · 24
    Ix91 · 26
    Ix92 · 26, 29, 31, 66, 110
    Ix94 · 25, 207
    Ix95 · 25, 207
    Ix96 · 25
    Ix97 · 36, 65
  VIs to access · 133

M

Memory
  VIs to access · 139

N

Numeric Data Types · 210

P

PLC
VIs to access · 182
Plotting
VIs to access · 188
PmacAcc · 4
PmacAccMachineInput8 · 5
PmacAccMachineOutput8 · 5
PmacAddress · 6
PmacAddressAdd · 6
PmacAddressDelete · 7
PmacAddressItem · 8
PmacAddressMotors · 7
PmacButton
PmacButtGetBool · 8
PmacButtGetDbl · 9
PmacButtGetLong · 9
PmacButtGetShort · 10
PmacButtGetStr · 10
PmacButtGetULong · 11
PmacButtGetUShort · 11
PmacButtSendStr · 11
PmacButtons · 8
PmacComm
PmacCommA ppend · 12
PmacCommBuffer · 12
PmacCommError · 13
PmacCommGetBuffer · 13
PmacCommGetStr · 14
PmacCommGlobal · 15
PmacCommLast · 16
PmacCommRespStr · 14
PmacCommSendStr · 14
PmacCoord · 16
PmacCoordColor · 16
PmacCoordCurrent · 17
PmacCoordDef · 17, 22
PmacCoordIVar · 17, 22
PmacCoordMotorDef · 18
PmacCoordMotorsToCoord · 18
PmacCoordMotorToCoord · 19
PmacCoordMotorToEncoder · 19
PmacCoordRunning · 20
PmacCoordScale · 20, 26
PmacCoordSpecify · 27
PmacCoordStat · 21
PmacCoordStatProg · 21, 28
PmacCoordStatWord1 · 21, 30
PmacCoordStatWord2 · 22, 33
PmacDevice · 36
PmacDevClose · 36
PmacDevSerial · 37
PmacDPR
PmacDPR · 37
PmacDPRFixedBack · 37
PmacDPRFixedBackConfig · 38
PmacDPRFixedBackCoord · 64
PmacDPRFixedBackCoordVector · 63
PmacDPRFixedBackMotorVector · 61
PmacDPRFixedBackVectors · 39
PmacDPRMotorVecToCoord · 40
PmacDPRNumericCINCluster · 40
PmacDPRNumericCluster · 41
PmacDPRNumericDbl · 41
PmacDPRNumericDWord · 42
PmacDPRNumericDWordBit · 43
PmacDPRNumericDWordBitTest · 44
PmacDPRNumericDWordSetMask · 44
PmacDPRNumericSpec · 45, 61
PmacDPRNumericWord · 45
PmacDPRRealTimeConfig · 46
PmacDPRRealTimeMotor · 47, 59
PmacDPRRealTimeMotors · 48
PmacDPRRealTimePVE · 49
PmacDPRRealTimeServo · 57
PmacDPRRealTimeVector · 56
PmacDPRRealTimeVectors · 49
PmacDPRServoDelta · 50
PmacDPRVarBack · 50
PmacDPRVarBackConfig · 51
PmacDPRVarBackgroundxLate · 52
PmacDPRVarBackReset · 53
PmacDPRVarBackSpec · 55
PmacDPRVarBackStat · 54
PmacDPRVarBackVectors · 53
PmacDPRFixedBackMotor · 62
PmacEncoder · 67
PmacEncoderCapture · 76
PmacEncoderCaptureControl · 85
PmacEncoderCaptureFlag · 84
PmacEncoderCompare · 76
PmacEncoderCompareConfig · 75, 84
PmacEncoderIVarCapture · 74, 82
PmacEncoderOffset · 74
PmacEncoderRegADC · 73
PmacEncoderRegCapture · 73
PmacEncoderRegDAC · 73
PmacEncoderRegisters · 72, 81
PmacEncoderRegServo · 72
PmacEncoderRegStat · 70
PmacEncoderRegTime · 70
PmacEncoderStatControl · 79
PmacEncoderStatFlags · 77
PmacEncoderToCoord · 69
PmacEncoderToEncoder · 68
PmacEncoderTrigger · 67
PmacFile · 86
PmacFileDatalog · 87
PmacFileDatalogAppend · 88
PmacFileDatalogControl · 92
PmacFileDatalogCreate · 88
PmacFileDatalogDisplay · 92
PmacFileDatalogRead · 89
PmacFileDownLoad · 89
PmacFileDownLoadLog · 90
PmacFileLoad · 90
PmacFileSave · 91
PmacFileSpreadsheet · 91
PmacGather · 93
PmacGatherCollect · 93
PmacGatherSelect · 94, 97
PmacGatherSetup · 95

PmacGatherSpec · 98
PmacGatherSpreadsheet · 95
PmacGatherStart · 95
PmacGatherStep · 96
PmacGatherStop · 96
PmacGlobal · 99
PmacGlobalBufferClose · 99
PmacGlobalBufferSize · 99
PmacGlobalControl · 100, 102
PmacGlobalIVarComm · 100, 106
PmacGlobalIVarMove · 100, 109
PmacGlobalStat · 101
PmacGlobalStatBuffer · 101, 111
PmacGlobalStatGather · 102, 112
PmacGlobalStatWord1 · 102, 113
PmacGlobalStatWord2 · 102, 115
PmacHome · 117
PmacHomeComplete · 118
PmacHomeIVar · 118
PmacHomeState · 117, 119
PmacHomeIVar · 121
PmacIVar
PmacIVarBool · 133
PmacIVarDbl · 134
PmacIVarGetBool · 134
PmacIVarGetDbl · 135
PmacIVarGetLong · 135
PmacIVarGetShort · 136
PmacIVarLong · 136
PmacIVarSetBool · 137
PmacIVarSetDbl · 137
PmacIVarSetLong · 137
PmacIVarSetShort · 138
PmacIVarShort · 138
PmacMemory · 139
PmacMemoryGet · 139
PmacMemoryGetBit · 140
PmacMemoryGetBits · 140
PmacMemoryRead · 140
PmacMemoryReadDbl · 141
PmacMemorySet · 141
PmacMemorySetBit · 142
PmacMemorySetBits · 142
PmacMemoryWrite · 142
PmacMemoryWriteDbl · 143
PmacMotor · 143
PmacMotorCurrent · 143
PmacMotorError · 144
PmacMotorIVarFlag · 144, 151
PmacMotorIVarMove · 145, 154
PmacMotorIVarPID · 146, 156
PmacMotorIVarSafety · 146, 161
PmacMotorJogControl · 147, 165
PmacMotorLimitControl · 147, 168
PmacMotorPosition · 148
PmacMotorPVE · 148, 170
PmacMotorStat · 149
PmacMotorStatJog · 149, 170
PmacMotorStatLimit · 149, 172
PmacMotorStatWord1 · 150, 174
PmacMotorStatWord2 · 150, 176
PmacMotorVelocity · 150
PmacMotors · 178
PmacMotorsCloseLoop · 178
PmacMotorsErrors · 179
PmacMotorsPlotSelect · 179, 181
PmacMotorsPositions · 180
PmacMotorsPVE · 181
PmacMotorsVelocities · 180
PMACPanel Organization
Device Management · 36
PmacPLC · 182
PmacPLCBuffer · 182
PmacPLCDownLoadBuffer · 183
PmacPLCEnable · 184
PmacPLCExec · 184, 187
PmacPLCList · 185
PmacPLCParse · 185
PmacPLCSelect · 185, 187
PmacPLCState · 186
PmacPLCVIState · 187
PmacPlot · 188
PmacPlotColors · 188
PmacPlotXYChartBuffer · 188
PmacPQM · 189
PmacPQMArray · 189
PmacPQMBool · 190, 195
PmacPQMBool2Var · 191
PmacPQMDbl · 191, 195
PmacPQMDbl2Var · 191
PmacPQMLong · 192, 196
PmacPQMLong2Var · 192
PmacPQMShort · 192, 196
PmacPQMShort2Var · 193
PmacPQMType · 197
PmacPQMVar2Bool · 193
PmacPQMVar2Dbl · 193
PmacPQMVar2Long · 194
PmacPQMVar2Short · 194
PmacPQMVariant · 194, 197
PmacProgram · 197
PmacProgBuffer · 198
PmacProgDebug · 198
PmacProgDownLoadBuffer · 199
PmacProgEdit · 200, 204
PmacProgExec · 200, 205
PmacProgJog · 201
PmacProgList · 202, 209
PmacProgParse · 202
PmacProgRun · 203
PmacProgSelect · 203
PmacProgVIState · 209
PmacResponse · 210
PmacRespGetBool · 210
PmacRespGetDbl · 210
PmacRespGetLong · 211
PmacRespGetShort · 211
PmacRespGetULong · 211
PmacRespGetUShort · 212
PmacSubVI · 212
PmacPLCSubVI · 212
PmacPLCSubVICreate · 213
PmacProgSubVI · 213
PmacProgSubVICreate · 214

PQM Variables
VIs to access · 189
Programs
VIs to access · 197, 212